DEMOCRATIC AND POPULAR REPUBLIC OF ALGERIA MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY OF SAAD DAHLAB BLIDA - 1

Faculty of Sciences

Department of Computer Science

Report submitted for the fulfilment of Master's degree

Domain: MI

Affiliation: Computer Science

Specialization: Computer systems and networks

Extending the Lifetime of Wireless Sensor Networks Using Variants of the N-Policy Vacation Policy

Abdelkader Soheib CHENNOUF and Mohamed Elhadi HADJER

Supervisor: Mrs. Bachira Boutoumi
Co-Supervisor: Mrs. Fatma Zohra Zahra
President: Mr. Mohamed Ould Khaoua
Examiner: Mrs. Imene Azouz

Academic year: 2024/2025

ملخص

الشبكات اللاسلكية لأجهزة الاستشعار ($WSNs$) تتكون من عقد استشعار موزعة مكانياًا، قادرة على مراقبة الظروف الفيزيائية أو البيئية مثل درجة الحرارة، الرطوبة، الحركة، أو الضغط، ونقل البيانات المجمعة إلى وحدة مركزية للتحليل. تلعب هذه الشبكات دوراً أساسياًا في مجالات متنوعة، بما في ذلك المراقبة البيئية، الزراعة الذكية، الأتمتة الصناعية، والأنظمة الصحية، حيث تكون جمع البيانات في الوقت الفعلي ضرورياًا لاتخاذ القرارات والتحكم في النظام.

على الرغم من فائدتها، تواجه الشبكات اللاسلكية لأجهزة الاستشعار قيوداً كبيرة بسبب الموارد الطاقية المحدودة لعقد الاستشعار، التي غالباًا ما تعمل ببطاريات محدودة وتوضع في بيئات يصعب الوصول إليها. هذا يجعل كفاءة الطاقة مصدر قلق رئيسي في تصميم الشبكات اللاسلكية لأجهزة الاستشعار.

في هذه الأطروحة، نقترح مجموعة من النماذج الرياضية لفهم وتحسين استهلاك الطاقة لعقد الاستشعار بشكل أفضل. يعتمد نهجنا على نظرية الطوابير ويدمج استخدام استراتيجية سياسة ، التي تؤخر الخدمة حتى يتم تجميع عدد عتبي من الحزم. نقوم بتقييم ومقارنة نماذج متعددة من خلال كل من الأساليب التحليلية باستخدام سلاسل ماركوف ذات الزمن المستمر والمحاكاة بالأحداث المنفصلة. تتيح هذه المنهجية المدمجة تقييم التسويات بين الطاقة والأداء تحت ظروف متنوعة. تساهم النتائج في تطوير استراتيجيات واعية بالطاقة وتقدم رؤى تدعم تصميم شبكات لاسلكية لأجهزة الاستشعار أكثر كفاءة واستدامة.

الكلمات المفتاحية: الكلمات المفتاحية: الشبكات اللاسلكية لأجهزة الاستشعار، الطوابير مع الإجازة ،سياسات إجازة ، سلاسل ماركوف ذات الزمن المستمر ($CTMCs$)، مقاييس الأداء.

Abstract

Wireless Sensor Networks (WSNs) are composed of spatially distributed sensor nodes capable of monitoring physical or environmental conditions such as temperature, humidity, motion, or pressure, and transmitting the collected data to a central unit for analysis. These networks play an essential role in various domains, including environmental monitoring, smart agriculture, industrial automation, and health systems, where real-time data collection is critical for decision making and system control.

Despite their usefulness, WSNs face significant limitations due to the constrained energy resources of sensor nodes, which often operate on limited battery power and are deployed in inaccessible environments. This makes energy efficiency a major concern in WSN design.

In this thesis, we propose a set of analytical models to better understand and optimize the energy consumption of sensor nodes. Our approach is based on queueing theory and incorporates the N-policy strategy, which delays service until a threshold number of packets has accumulated.

We evaluate and compare several models using both analytical techniques based on Continuous-Time Markov Chains and discrete-event simulation. This combined methodology enables a comprehensive evaluation of energy performance trade-offs under different system conditions. The results contribute to the development of energy aware strategies and offer insights for designing more efficient and sustainable wireless sensor networks.

Keywords: Wireless Sensor Networks, Queues with Vacation, Vacation policies, Continuous-Time Markov Chains (CTMCs), Performance Metrics.

Résumé

Les réseaux de capteurs sans fil (WSNs) sont composés de nœuds capteurs distribués spatialement, capables de surveiller des conditions physiques ou environnementales telles que la température, l'humidité, le mouvement ou la pression, et de transmettre les données collectées à une unité centrale pour analyse. Ces réseaux jouent un rôle essentiel dans divers domaines, notamment la surveillance environnementale, l'agriculture intelligente, l'automatisation industrielle et les systèmes de santé, où la collecte de données en temps réel est cruciale pour la prise de décision et le contrôle des systèmes.

Malgré leur utilité, les WSNs présentent des limitations importantes dues à la contrainte énergétique des nœuds capteurs, qui fonctionnent souvent sur batterie et sont déployés dans des environnements difficiles d'accès. Cela rend l'efficacité énergétique un enjeu majeur dans la conception des WSNs.

Dans ce mémoire, nous proposons un ensemble de modèles mathématiques visant à mieux comprendre et optimiser la consommation énergétique des nœuds capteurs. Notre approche repose sur la théorie des files d'attente, à travers laquelle nous évaluons et comparons différents modèles afin d'identifier des stratégies qui équilibrent économies d'énergie et réactivité du système. Ce travail contribue au développement de solutions économes en énergie et apporte des perspectives utiles pour la conception de réseaux de capteurs plus efficaces et durables.

Mots-clés : Réseaux de capteurs sans fil, Files d'attente avec vacances, Politique de vacances, Chaînes de Markov en temps continu (CTMC), Indicateurs de performance.

Acknowledgements

All praise is due to Allah, the Most Gracious and the Most Merciful. I am deeply grateful to Him for granting me the strength, patience, and perseverance to complete this thesis. Without His guidance and blessings, this work would not have been possible.

I would like to extend my sincere gratitude to my supervisor Mrs. BACHIRA BOUTOUMI, for her continuous support, encouragement, and insightful supervision throughout the course of this research. Their guidance has been invaluable both academically and personally.

We would like to express our sincere gratitude to our supervisor, Mrs. Bachira Boutoumi, for her unwavering support, valuable guidance, and continuous encouragement throughout the course of this thesis. Her deep insight and experience in the field have been instrumental in shaping our work and helping us overcome challenges with clarity and confidence. We are equally grateful to our co-supervisor, Mrs. Fatma Zohra Zahra, whose constructive feedback and thoughtful suggestions have greatly enriched the quality of this research.

We extend our heartfelt appreciation to the members of the thesis jury for their time and effort in evaluating our work. We are honored by the presence of Mr. Mohamed Ould Khaoua, the President of the Jury, and Mrs. Imene Azouz, our Examiner. Their insightful remarks and professional perspectives are crucial in enhancing the rigor and relevance of our study.

we also wish to thank the faculty and staff of the Department of Computer science at Saad Dahleb University for their assistance and for providing an environment conducive to research and learning,To our colleagues and friends, thank you for your support, motivation, and collaboration. The discussions, shared challenges, and mutual encouragement helped make this journey much more manageable and enjoyable.

Abdelkader Soheib CHENNOUF
Mohamed Elhadi HADJER

To my beloved mother, whose love has been my anchor and whose quiet strength has guided me through every season of life. Thank you for every morning you drove me to school as a child, for every cup of coffee you prepared with care, and for every time you stood by me without hesitation. Your presence has been my comfort, your support my strength, and your faith in me my greatest motivation. This journey would not have been possible without you.

In memory of my father, I dedicate this work to the memory of my beloved father, **Chennouf Tarik**, who passed away on feburary 2024, whose presence still echoes in every step I take. He was not only my father, but my mentor, my first friend, my confident, and the spark that ignited my love for engineering. He believed in me before I even believed in myself—spoke of me with pride to friends, family, and colleagues, as if his own soul was intertwined with my ambitions. He taught me the sacred values of loyalty, virtue, courage, hard work, and discipline, not with words, but with the example of his life. Each morning I wake, each time I start my engine, each line I write on this very laptop he gave me, I remember him. He is in my voice, my smile, my silhouette. He taught me how to drive, how to think, how to rise after failure. He gave me my name, and even now, when I look in the mirror, I see his reflection looking back. How can one say goodbye to someone who never truly left? He is in every line of this thesis, in every achievement I make, and in every heartbeat that pushes me forward. This work is for him—*my eternal teacher, my father*.

Abdelkader Soheib CHENNOUF

On a more personal note, I would like to express my infinite gratitude to the person who has shaped the foundation of who I am my father. Through his unwavering dedication, sacrifices, and quiet strength, he showed me what it means to work hard, live with integrity, and always stand tall. His journey, full of challenges and resilience, is a constant reminder of the values I carry with me today.

To my mother, the soul of our home and the tireless heart behind every small and great moment no words can ever do justice to your boundless love and the sacrifices you've made. You gave so much of yourself so we could stand where we are today, and for that, I will be forever grateful.

To my two sisters and two brothers, thank you for walking beside me through the storms and the sunshine. Your love, laughter, and unwavering belief in me have been a comforting presence in every stage of my life.

I am especially thankful to my aunts, who have always cheered for me from the sidelines with heartfelt encouragement, generous support, and unconditional affection. Your words and presence mattered more than you know.

And to my beloved grandmother, who left this world two years ago but never left my heart this work is dedicated to you. Your wisdom, your stories, and your gentle hands still guide me. I hope I've made you proud.

To all the friends and colleagues I've met throughout these years thank you. You were my second family. From classroom stress to late night study sessions and shared laughter, you made this journey not just bearable, but memorable.

Mohamed Elhadi HADJER

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

WSN	Wireless Sensor Network
QoS	Quality of Service
IEEE	Institute of Electrical and Electronics Engineers
MAC	Medium Access Control
CTMC	Continuous-Time Markov Chains
DTMC	Discrete Time Markov Chain
GSPN	Generalized Stochastic Petri Net
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
DES	Discrete-Event Simulation
DSME	Deterministic and Synchronous Multichannel Extension
EHP	Energy Hole Problem
FEL	Future Event List
GTS	Guaranteed Time Slot
GreatSPN	Graphical Editor and Analyzer for Timed and Stochastic Petri Nets
MAPE	Mean Absolute Percentage Error
sMAPE	Symmetric Mean Absolute Percentage Error
TSCH	Time Slotted Channel Hopping
ASN	Absolute Slot Number
CFP	Contention-Free Period
CAP	Contention Access Period
SO	Superframe Order
BO	Beacon Order
MO	Multi-superframe Order
MR-WSN	Multi-Radio Wireless Sensor Network
PSO	Particle Swarm Optimization
PCB	Printed Circuit Board

General Introduction

Wireless Sensor Networks (WSNs) have emerged as a fundamental enabling technology in numerous domains due to their ability to autonomously monitor, sense, and communicate environmental conditions in real time. Composed of spatially distributed sensor nodes equipped with limited power sources, WSNs are now essential in applications such as environmental or industrial monitoring, military surveillance, healthcare systems, precision agriculture, and smart infrastructure [1]. Their capability to operate in remote or hazardous environments and provide data-driven insights has significantly advanced the field of distributed sensing and real-time decision making.

A major challenge in WSNs is the limited energy capacity of sensor nodes, which are often deployed in remote or inaccessible areas, making battery replacement or recharging impractical. A significant portion of energy is consumed during frequent transitions between idle and active states. As stated in [2], "*switching from idle to busy state and vice versa takes up a major portion of the power*". To address this, threshold vacation policies like the N-policy delays activation until a buffer threshold is reached, reducing the number of transitions. However, this introduces latency [3]. Minimizing transitions reduces energy consumption but at the cost of increased packet waiting time. Optimizing this tradeoff is critical, as it directly impacts both network lifetime and service responsiveness.

In examining prior N-policy-based models for WSNs as a solution for the previous problem, several limitations stood out. Some assume infinite buffer capacity [4], which does not reflect the hardware constraints of real sensor nodes. Others model only basic idle-active transitions [2]. Additionally, while batch processing has been explored, it is often introduced without formal analysis under realistic queueing assumptions such as finite capacity [5]. Observing these gaps has motivated us to investigate more accurate modeling approaches that address these constraints from the outset.

In this work, we propose several queueing models for wireless sensor nodes that integrate batch service with a service-hold. Some models include only one of these ideas, while

others combine both. We believe that batch transmission can reduce per-packet overhead, and that service-hold can lower the frequency of costly wake-up transitions by avoiding an immediate return to idle when the buffer is emptied. The remainder of this work is dedicated to evaluating whether these policies truly contribute to reducing energy consumption and improving latency under realistic system constraints.

Structure of the report

- **Chapter 1: State of the Art**
  Presents background on WSNs, queuing theory, Markov chains, simulation, and existing N-policy models for energy-delay optimization and presents some related work for our problematic.

- **Chapter 2: Modeling of the Sensor Node System**
  Introduces our proposed models, simulation assumptions, and formalizes several queuing-vacations policies for managing node behavior.

- **Chapter 3: Implementation, Experimental Results, and Discussion**
  Describes the simulation setup, We describe the hardware/software setup, graphical interfaces, A comprehensive set of experiments is then conducted to evaluate the impact of system parameters on energy consumption, delay, drop rate, and activation rate. The results are analyzed to highlight trade-offs and guide design decisions.

Chapter 1

State of the Art

1.1. Introduction

This chapter outlines the essential concepts and prior research needed to support the modeling and evaluation of energy efficient mechanisms in Wireless Sensor Networks (WSNs). It serves as a foundation for understanding the tools used in this thesis, including queueing theory, Continuous-Time Markov Chains (CTMCs), and discrete-event simulation (DES). These concepts are necessary for analyzing the trade-off between energy and delay and for justifying the use of sleep scheduling strategies like duty cycling and threshold-based policies

1.2. An Overview of WSNs

WSNs represent a key enabling technology in embedded systems, offering scalable and energy efficient solutions for collecting and transmitting data in diverse and often resource constrained environments. A WSN is composed of spatially distributed sensor nodes that cooperate to monitor physical or environmental parameters such as temperature, humidity, sound, or motion. These nodes communicate wirelessly, typically over short ranges, and transmit their readings to a central base station or sink node for processing and analysis [1,6].

1.2.1. Components of a Sensor Node

Each wireless sensor node is typically compact in size and composed of several integrated components:

Sensing Unit: This unit interacts with the physical environment to monitor parameters such as temperature, humidity, vibration, or gas levels. It converts physical stimuli into electrical signals through appropriate sensors [7, Section 2.1.5, pp. 31–32].

Processing Unit: At the heart of the node is a microcontroller, which processes sensor data, executes application logic, and manages communication protocols [7, Section 2.1.2, pp. 19–21].

Communication Unit: This component provides wireless connectivity by transmitting and receiving data to and from neighboring nodes or a central base station [7, Section 2.1.4, pp. 21–30].

Power Unit: Sensor nodes are generally powered by batteries, which may be supplemented by energy harvesting components such as solar panels or vibration-based generators [7, Section 2.1.6, pp. 32–36].

These components operate in a tightly integrated manner to perform sensing, local data processing, wireless communication, and energy management, forming the backbone of typical WSN functionality. The overall structure of a sensor node, as illustrated in figure 1.1, which is adapted from [8].

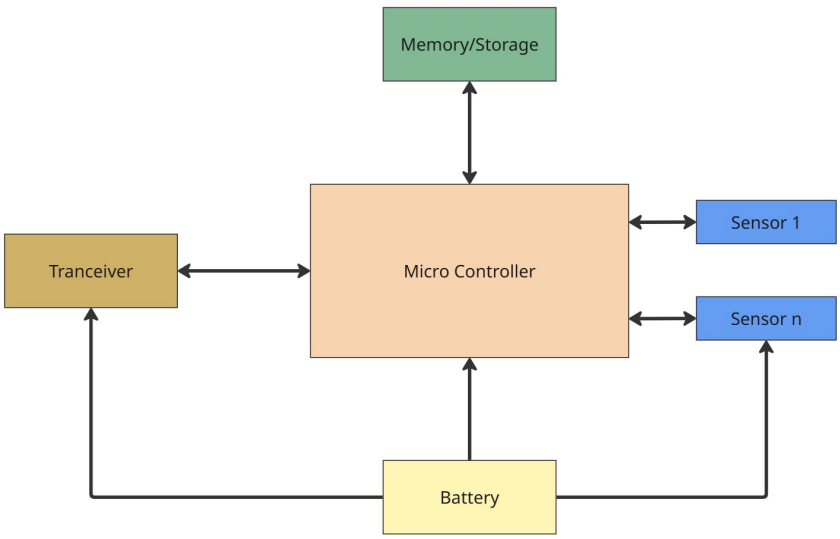

Figure 1.1: Sensor node components

1.2.2. Applications of WSNs

WSNs have demonstrated significant versatility across various domains due to their ability to operate autonomously, adapt to different environments, and scale efficiently. Their deployment spans numerous fields where real-time data acquisition is essential [1]:

- **Environmental applications:** WSNs support environmental monitoring tasks such as forest fire detection, flood warning, biocomplexity mapping, and precision agriculture through dense, unattended sensor deployment [1, Section 2.2].
- **Health applications:** WSNs enable monitoring of physiological data, patient and doctor tracking, and safer drug administration. [1, Section 2.3].
- **Commercial applications:** WSNs enable automation, tracking, and control in commercial settings like inventory management, smart buildings, and vehicle monitoring [1, Section 2.5].
- **Military applications:** WSNs enhance battlefield awareness through surveillance, troop monitoring, targeting, and NBC threat detection [1, Section 2.1].

1.2.3. WSNs Architecture and Communication

WSNs often follow a hierarchical architecture comprising three key components: sensor nodes, cluster heads (or sink nodes), and a central base station. Sensor nodes are responsible for data collection and short range communication, while cluster heads aggregate this data and forward it to the base station for centralized processing. To reduce energy consumption, WSNs typically use a multi-hop communication model where data is transmitted through intermediate nodes rather than directly over long distances [9]. The overall architecture, including these roles, is illustrated in figure1.2, which is adapted from the original diagram presented in [9, Figure 1, p. 3], reflecting the hierarchical and cooperative structure typical of WSN communication.

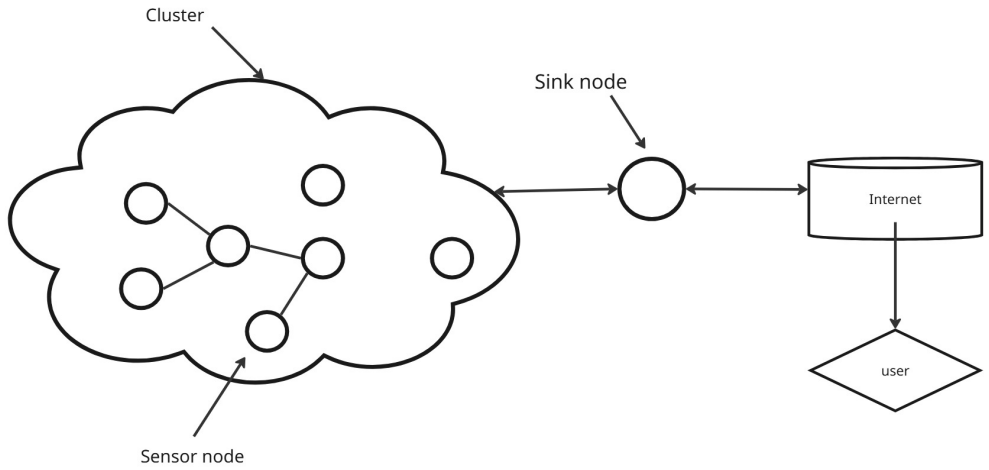

Figure 1.2: Basic WSNs architecture

1.2.4. Common Topologies in WSNs

The performance, scalability, and energy efficiency of a WSN are largely determined by its underlying topology. Different application scenarios and deployment environments require different organizational structures. The most commonly employed WSN topologies include [10]:

- **Star Topology:** In this configuration, all sensor nodes communicate directly with a central sink or coordinator node. It is simple and energy efficient due to single hop communication, but suffers from a single point of failure if the central node becomes unavailable.
- **Tree Topology:** This topology organizes nodes in a hierarchical manner, resembling a tree. Data is forwarded from leaf nodes up through parent nodes toward the sink. It is scalable and supports structured routing, but the hierarchical paths may increase latency and create dependency bottlenecks.
- **Mesh Topology:** In a mesh setup, each node can connect with multiple neighboring nodes, enabling multi-hop communication and multiple routing paths. This enhances fault tolerance and reliability, but can increase energy consumption and routing complexity.
- **Ring Topology:** Nodes are deployed in a ring-like formation, with each node typically forwarding data to the nearest neighbor until it reaches the sink. This structure balances communication load and is easy to maintain, but may suffer from latency if the path is long.
- **Hybrid Topology:** A hybrid topology combines features of two or more standard topologies to exploit their individual strengths. For example, a network may implement star topologies within local clusters, interconnected through a mesh or tree backbone. This offers adaptability to varying application demands and physical layouts.

An illustrative comparison of these common WSN topologies is presented in Figure 1.3, showing their typical structural layouts and node organization.

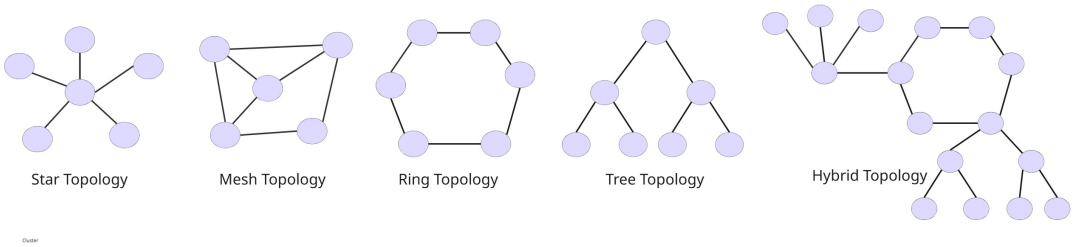

Figure 1.3: Common WSNs topologies: star, tree, mesh, ring and hybrid.

1.2.5. The energy-delay tradeoff problem

In WSNs, two performance objectives dominate design decisions: conserving energy and minimizing communication delay. Unfortunately, these goals are often in direct conflict. Enhancing energy efficiency typically involves techniques such as sleep scheduling and duty cycling, which can delay communication. Conversely, reducing delay demands immediate data transmission, which consumes more power. This tradeoff lies at the core of nearly every WSNs protocol and architecture [11].

Sensor nodes are generally powered by batteries and often deployed in environments where maintenance is infeasible. Hence, prolonging network lifetime is paramount. According to the IEEE 802.15.4 standard, the most significant contributor to energy depletion is radio communication, not sensing or processing [12]. Transmitting even a few bits of data can consume magnitudes more energy than performing a local computation. Therefore, many energy-aware designs focus on minimizing radio activity duration and frequency through duty-cycling or data aggregation [13, 14].

Figure 1.4 presented in [15] illustrates the typical power consumption distribution among the main subsystems of a sensor node, highlighting how the radio transceiver dominates energy usage compared to sensing and processing components.

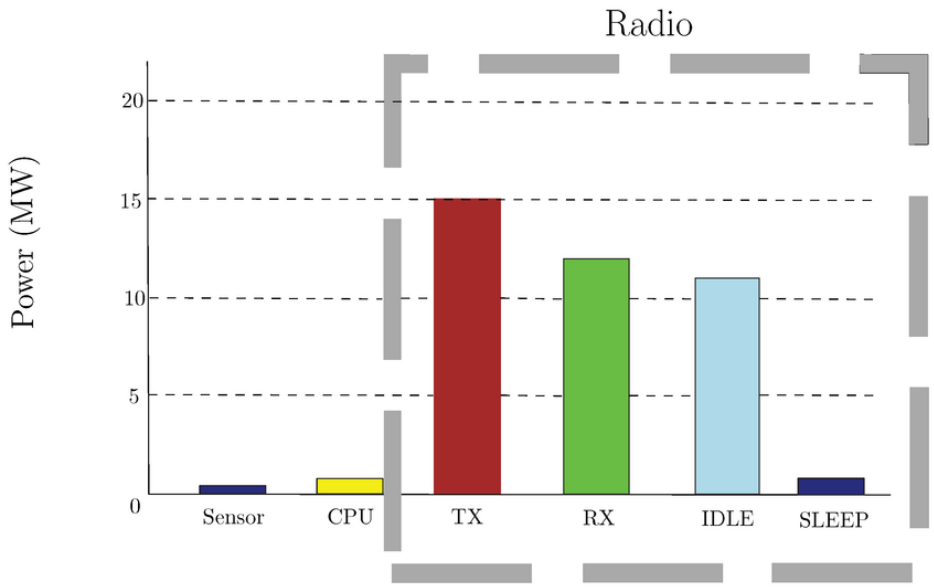

Figure 1.4: Energy consumption in WSNs

1.2.6. Power Saving in WSNs

Over the years, various energy-saving mechanisms have been proposed across different layers of the network stack. In this section, we focus specifically on duty cycling.

Duty Cycle Approach: is a widely used energy-saving mechanism in WSNs, where sensor nodes periodically switch between active and sleep modes to reduce power consumption. Rather than immediately reacting to each data arrival, these transitions are governed by a predefined sleep wake control algorithm that determines the optimal timing for activating or deactivating the radio. This logic can incorporate traffic conditions, energy constraints, activation policies or application-level requirements. The proportion of time a node remains in the active state is referred to as its duty cycle, and optimizing it plays a key role in extending network lifetime without compromising performance [16, Section 3.1].

Other Techniques: In addition to duty cycling, several other energy saving approaches have been proposed to enhance the efficiency of WSNs. These include routing [16, Section 3.2], energy efficient MAC protocols [16, Section 3.3], data aggregation [16, Section 3.4], cross layer design [16, Section 3.5], and error control code [16, Section 3.6].

1.3. Queueing Theory with vacation

Queueing theory is a branch of applied probability used to study the behavior of waiting lines across a wide range of real-world systems. It provides mathematical tools for analyzing

situations in which entities arrive to receive a service that may not be immediately available, leading to queues or delays [17, Section 1.1, p. 1–2]. This chapter introduces the basic concepts of queueing theory that are relevant to our work, with particular emphasis on vacation-based models, which form the core of our study.

1.3.1. Characteristics and Notation of Queueing Systems

A queueing system is defined by a set of fundamental characteristics that describe its operation and performance. The most commonly used classification includes the following elements [17, Section 1.2, pp. 4–6]:

Arrival Process: This defines how entities arrive at the system. It is typically modeled as a stochastic process, with interarrival times often following a known probability distribution. Common considerations include whether arrivals occur individually or in batches, and whether the arrival pattern is stationary (constant over time) or nonstationary (time-dependent).

Service Process: Service times are generally random and described by a probability distribution. The model may involve individual or batch service, and in some cases, the service rate may depend on the number of entities in the queue (state-dependent service). Like arrivals, service can be stationary or vary with time due to factors such as learning effects or resource constraints.

Number of Servers: This represents the number of parallel service channels available in the system. Increasing the number of servers can reduce delays but increases cost. Queueing systems may be organized with a single queue feeding all servers or multiple queues for each server.

System Capacity: This refers to the maximum number of entities that the system can accommodate at once, including both those waiting and those being served. If the capacity is limited, new arrivals may be blocked or turned away once the system is full.

Queue Discipline: The service order is determined by the queue discipline. Typical rules include First-Come-First-Served (FCFS), Last-Come-First-Served (LCFS), round robing, processor sharing , and various priority-based schemes, which may be preemptive or non-preemptive.

Kendall's notation: Queueing models are commonly described using Kendall's notation: A/B/X/Y/Z, where each symbol characterizes a fundamental property of the system [17, Section 1.2.7, p. 7–8]. Specifically, A denotes the distribution of arrival interarrival times, B the distribution of service times, X the number of parallel servers, Y the system capacity, and Z the queue discipline.

For instance, a system denoted as M/M/1 represents a single-server queue with Poisson arrivals and exponential service times [17, Section 3.2, pp. 77–78]. To visualize the structure of such a system, figure 1.5 illustrates a basic M/M/1/K queue [18, Section 3.6], showing the flow of entities through a finite-capacity queue, entities arrive according to a Poisson process (λ) and are served exponentially (μ) by a single server.

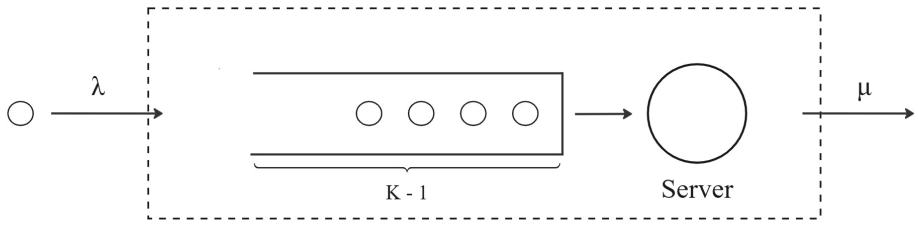

Figure 1.5: Diagram of an M/M/1/K queue.

1.3.2. Vacation Queueing models

In many real-world queueing systems, servers are not always continuously available, as assumed in classical models. Instead, they may become temporarily unavailable, referred to as taking vacations to perform other tasks, undergo maintenance, or simply pause when there is no immediate work. Introducing vacation periods into queueing models adds flexibility and realism, making them better suited for analyzing systems where service availability fluctuates. This perspective has gained prominence since the 1970s with the rise of complex and technology-driven service systems, where understanding performance under intermittent availability is crucial [19, Section 1.2, pp. 1–2].

Key components of vacation queueing models: Vacations models are defined by three key components: the rule that initiates the vacation, the condition for resuming service, and the statistical distribution of the vacation duration [19, Section 1.2, pp. 3–4].

- **Vacation Initiation Rule:** This rule defines when the server may begin a vacation. In exhaustive systems, vacations start only when the system is empty. Non-exhaustive

systems allow vacations even with waiting entities. In multi-server settings, semi-exhaustive rules may let only some servers take vacations. Unexpected service interruptions can also trigger a vacation.

- **Vacation Termination Rule:** Termination rules determine when the server returns to service. In a single vacation policy, one vacation follows each busy period. Under a multiple vacation policy, the server continues to take vacations until it finds customers waiting. More complex rules, like threshold policies, adapt the return based on system state.
- **Vacation Duration Modeling:** Vacations are typically modeled as random variables with independent and identical distribution. Some models use a single distribution for all vacations, while others allow multiple types with distinct statistical properties, depending on the application or system behavior.

Threshhold policies: Several threshold-based service control strategies have been developed to optimize server activation timing under varying operational constraints.

- **N-Policy:** This threshold-based mechanism keeps the server in a vacation or idle state until the number of accumulated customers reaches a predefined threshold N. Only then does the server begin a new busy period to serve the queue. This policy is commonly used in systems with significant setup costs or power constraints, as it reduces the number of activations by batching service sessions, offering an effective balance between operational efficiency and service delay.
- **Other Threshold Policies:** Several related policies build upon the threshold concept. In the T-policy , the server takes a vacation of fixed duration T after each busy period and resumes service only if customers are present at the end of that interval [20]. The D-policy delays service until the total required workload of queued customers exceeds a predefined threshold D [21]. Finally, the Min(N,T) policy triggers service when either N customers have arrived or a maximum waiting time T has elapsed, whichever occurs first. [3]

1.4. Continuous-Time Markov Chains as a Modeling and Performance Evaluation Tool

This section presents the key elements of CTMCs as used in system modeling. It covers the core concepts, including state diagrams, the infinitesimal generator matrix, and steady-state analysis for evaluating long-term performance.

1.4.1. Continuous-Time Markov Chains (CTMCs)

A CTMC is a stochastic process where state transitions occur at arbitrary points in continuous time, governed by the Markov property which means that the future state depends only on the present state and not on the sequence of past events. Unlike DTMCs, CTMCs model systems assume that the time spent in each state follows an exponential distribution, making them particularly suited for representing real-world systems like WSNs where events unfold asynchronously. Time-homogeneous CTMCs assume constant transition rates over time and are characterized by an infinitesimal generator matrix, which can be used to formulate both differential equations for analyzing transient behavior and systems of linear algebraic equations for computing steady-state distributions. In this work, we focus on the steady-state approach, which enables the derivation of long-term performance metrics [22, Section 2.1.2.2, pp. 49–55].

State Diagram: The structure of a CTMC can be effectively represented using a state diagram (also known as a transition diagram). In this directed graph, each node corresponds to a state of the system, and each directed arc between nodes represents a possible transition, labeled with the corresponding transition rate q_{ij}. Transitions are included in the diagram only when the associated rate is nonzero. Self transitions are typically omitted visually, as they are implicitly encoded in the diagonal elements of the infinitesimal generator matrix. The diagram aids system visualization and analysis [22, 23]. An example of such a diagram is illustrated in Figure 1.6.

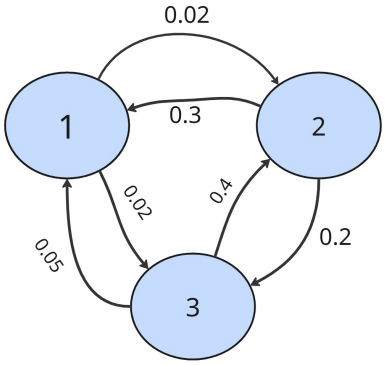

Figure 1.6: Example of a state diagram for a 3-state CTMC

Infinitesimal Generator Matrix: The infinitesimal generator matrix $Q = [q_{ij}]$ captures the transition dynamics of a CTMC. Each off-diagonal element q_{ij} (for $i \neq j$) represents the transition rate at which the process moves from state i to state j, as defined by the structure and parameters of the modeled system. These rates are typically extracted directly from

the transition diagram or known system behavior. The diagonal elements q_{ii} are computed to ensure that each row of the matrix sums to zero, satisfying the probability conservation property inherent to CTMCs [23, Chapter 3, pp. 25–32], [22, Section 2.1.2.2, pp. 49–55], The matrix shown in equation 1.1 corresponds to the infinitesimal generator of the CTMC illustrated in figure 1.6.

$$Q = \begin{bmatrix} -0.04 & 0.02 & 0.02 \\ 0.3 & -0.5 & 0.2 \\ 0.05 & 0.4 & -0.45 \end{bmatrix} \tag{1.1}$$

Steady-State Solution: To analyze the long-term behavior of the system, we solve for the stationary distribution $\boldsymbol{\pi}$, where each element π_i represents the probability of being in state i. The stationary distribution satisfies the following equations 1.2, as described in [22, Chapter 5, p. 167]:

$$\boldsymbol{\pi} \times Q = \mathbf{0}, \quad \text{with} \quad \sum_{i=1}^{n} \pi_i = 1 \tag{1.2}$$

1.4.2. Performance Metrics via Markov Chains

Once a system is modeled using a Markov chain, several performance indicators can be derived to assess its behavior over time. These metrics are essential for evaluating system efficiency, congestion, and stability.

System Utilization: Utilization refers to the fraction of time the system's resource (e.g., server or processor) is actively in use. If states S_b denote busy states, utilization ρ is computed as shown in 1.3 according to [22, Chapter 6, Eq. (6.3), p. 213]:

$$\rho = \sum_{i \in S_b} \pi_i \tag{1.3}$$

Average Number in System: Let n_i denote the number of entities in state i. The average number of entities in the system shown in 1.4 is defined in [22, Chapter 6, Eq. (6.8), p. 214]:

$$\mathbb{E}[N] = \sum_{i} n_i \pi_i \tag{1.4}$$

Throughput: Throughput refers to the average number of tasks completed per unit time. For CTMCs, it can be estimated by combining transition rates with steady-state probabilities

discussed in [22, Chapter 10, Eq. (10.147), p. 509] and shown in 1.5:

$$\text{Throughput} = \sum_{i,j} \pi_i q_{ij} \tag{1.5}$$

Average Sojourn Time: The sojourn time represents the average time an entity spends in the system. It can be determined using Little's Law appears in [22, Chapter 6, Eq. (6.9), p. 214]. In this expression 1.6, λ denotes the effective arrival rate:

$$\mathbb{E}[T] = \frac{\mathbb{E}[N]}{\overline{\lambda}} \tag{1.6}$$

1.5. Discrete-Event Simulation

Simulation is a key technique for studying systems that are too complex or impractical to analyze analytically. It is widely used in fields such as engineering, operations research, and network performance evaluation. Simulation systems can generally be categorized as either discrete or continuous, depending on whether their state variables change at specific points in time or continuously. As noted in [24, Section 1.7, p. 9], most real-world systems exhibit both types of behavior, but typically one dominates enough to classify the system as either discrete or continuous. This section focuses on DES, a modeling approach particularly suited to systems in which state transitions are triggered by distinct events, such as in WSNs.

1.5.1. Overview of DES

DES models systems in which state changes occur at distinct, instantaneous events over time. Instead of advancing time continuously, the simulation jumps from one scheduled event to the next [24, Section 1.7, p. 9]. A typical DES model consists of [24, Section 3.1, p. 61]:

- **Entity:** Any object or component in the system (e.g., a packet, a customer).
- **Event:** A Trigger that cause state transition (e.g., arrival, departure).
- **Activity:** Time-consuming process with known duration (e.g., service time).
- **Future Event List:** stores all scheduled events ordered by their time of occurrence.
- **Delay:** Waiting periods with unknown or conditional duration (e.g., waiting in queue).
- **System state:** A collection of variables describing the system at a given time.
- **Clock:** A variable used to represent simulated time.

Event Scheduling: In DES, the simulation loop operates using an event scheduling (or time-advance) approach, where the simulation clock progresses by jumping directly to the time of the next scheduled event. Events are managed through the FEL, which maintains upcoming events in chronological order. At each step, the earliest event is removed from the FEL, the simulation clock is updated accordingly, the event logic is executed to update the system state, and any resulting future events are scheduled by inserting them into the FEL. This strategy enhances computational efficiency by avoiding unnecessary time increments [24, Section 3.1, pp. 61–66], figure 1.7 presented in [25] illustrates event scheduling in simulation flowchart algorithm.

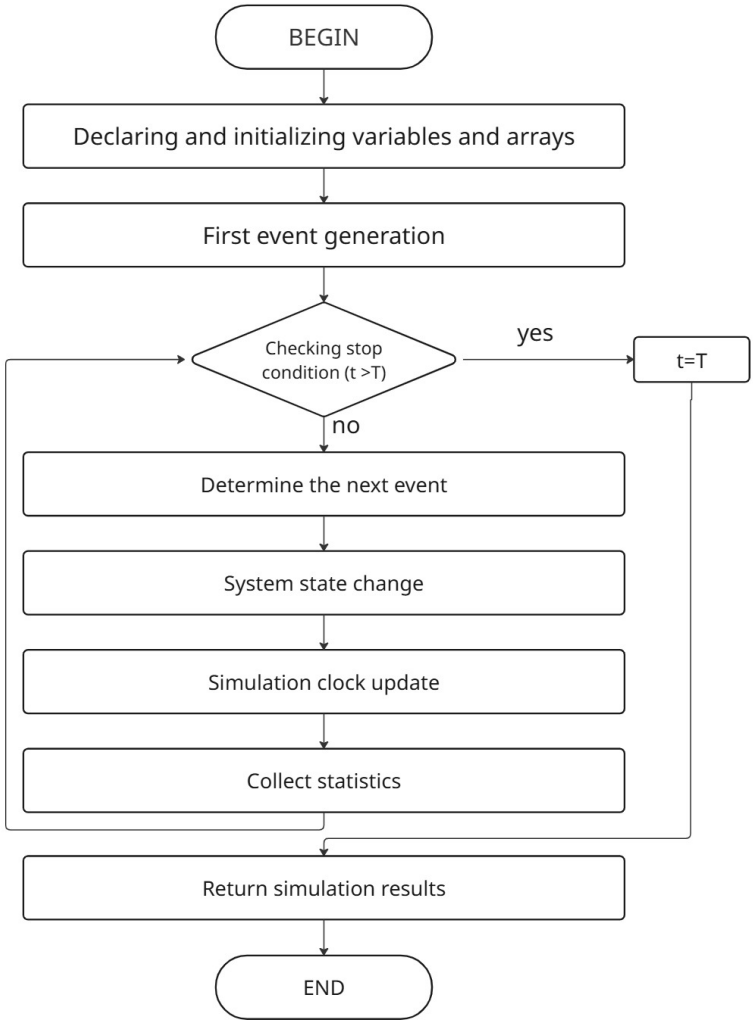

Figure 1.7: Flowchart of a DES model.

1.5.2. Steady-State Data Collection in Simulation

In simulation-based performance evaluation, distinguishing between the transient and steady-state phases is critical. During the initial transient phase, the system's behavior may be influenced by arbitrary starting conditions, potentially introducing bias into performance metrics. To mitigate this, a warm up period is often used, during which early simulation data are discarded to allow the system to stabilize. One common approach for collecting steady-state performance data is the Replication Method. This method involves executing multiple simulation runs using independent random number streams, discarding data collected during the warm-up phase, and computing statistics over the remaining data to estimate long-run behavior. Other standard techniques include the batch means method and time-averaged sampling, which divide the post-transient output into segments to compute more reliable confidence intervals for steady-state estimates [24, Section 11.5, pp. 352–370].

1.6. Queueing Models and N-Policy for Energy-Delay Trade-off

Energy efficiency in WSNs has been a major research focus, with numerous studies proposing queue-based models to optimize node power consumption. Researchers have explored these models and techniques under different network conditions and constraints, leading to a rich body of work that addresses the fundamental trade-offs between energy savings and system performance. From these, we can cite:

- **Chen et al. (2017)** [26] developed an adaptive N-policy queueing system for WSNs to address the critical energy-delay tradeoff, dynamically adjusting the queue threshold (N) based on real-time traffic conditions and challenging the traditional view that higher N-values always yield better energy savings. Their system operates through two key states: a sleep state with minimal energy consumption and no packet processing, and an active state where nodes process packets upon reaching the adaptive threshold N. The study introduced novel analytical models for determining optimal N-values under varying traffic loads, with simulations using practical parameters (e.g., 24.75mW active vs 0.015mW sleep power) demonstrating 15–30% energy reduction compared to fixed N-policy systems, consistent adherence to strict delay requirements (0.2s upper bound), and superior performance to conventional duty cycle approaches in both power efficiency and latency.

- **Goswami and Mund (2022)** [5] proposed a modified N-policy queueing model to prolong the lifetime of WSNs by introducing three operational states for sensor nodes:

sleep state (minimal energy consumption with no packet processing), idle state (waiting for packets to accumulate until reaching threshold N), and busy state (processing packets in batch mode for the first N packets followed by single-service mode). Their model combines batch processing of initial packets with individual service for subsequent arrivals to reduce both energy consumption and packet waiting times, featuring infinite buffer capacity with Poisson arrivals, dual service rates (μ_1 for single packets and μ_2 for batches where $\mu_2 < \mu_1$), and automatic transition to sleep mode when the queue empties. Through numerical analysis with parameters ($\mu = 5.0$, $\lambda = 3.0$, $\alpha = 0.7$), the study demonstrated 15–20% reduction in power consumption per cycle compared to standard N-policy, increased system lifetime through optimized threshold selection (testing $N = 2$–10), and better performance than conventional models in queue length and processing delay, while revealing the inherent trade-off between waiting time (increasing with N) and energy efficiency. The modified policy proved effective for balancing energy savings with service quality, though requiring more complex hardware implementation, and provided practical guidelines for threshold selection based on traffic patterns.

- **Zhang, Yang, and Wang (2023)** [27] proposed an adaptive N-policy sleep scheduling model to improve the timely reliability and energy efficiency of heterogeneous WSNs. Unlike conventional N-policy strategies, their model introduces a wait state between the work and sleep states. When the node becomes idle, it temporarily enters this wait state instead of switching to sleep immediately. If packets arrive during this short wait period, the node resumes processing without transitioning to sleep, reducing both delay and energy consumed through frequent state switching. The queueing system is modeled using an N-policy M/M/1/C queue with exponential waiting time, incorporating node heterogeneity in terms of processing capacity and rate. Performance metrics such as energy consumption, average delay, and timely reliability are derived analytically and evaluated via Monte Carlo simulations. The proposed strategy demonstrated up to 31.3

- **Jiang, Huang, Yang, and Leu (2012)** [28] introduced a queue based N-policy M/M/1 model to extend the lifetime of WSNs by addressing the energy hole problem (EHP) near the sink node. In their model, each sensor node activates its radio transmitter only when a queue of data packets reaches a predefined threshold N, minimizing energy wasted from frequent radio state transitions. Two main operational states are defined: idle (radio off, fewer than N packets) and busy (radio on, continuous service until queue is empty). The model assumes Poisson arrivals and exponential service times, with infinite buffer capacity. Analytical expressions for power consumption and queue behavior were derived, leading to a total expected power function optimized by tuning

N. Simulation results using MATLAB and NS-2 confirmed that, with optimal N, power consumption can be reduced by up to 28.79

- **Ghosh and Unnikrishnan (2019)** [3] examined an N-policy M/M/1 queuing model to address power consumption and delay in WSNs. Their approach activates the radio transmitter of a sensor node only when the queue length exceeds a threshold N, thereby minimizing power wasted from frequent transitions between active and idle modes. The system operates in two main states: idle (radio off when queue $< N$) and busy (radio on until queue empties). The authors extended this to a Min(N,T) policy, introducing a timer T to trigger transmission if packets are delayed too long, ensuring timely data delivery even during sparse traffic. The analytical model incorporates Poisson arrivals, exponential service times, and a Markovian queuing framework. Performance metrics such as average power consumption, queue length, and waiting time were derived mathematically and validated via simulations using MATLAB. The study found that optimal N values shift with arrival rates (e.g., $N = 4$ for $\lambda = 1$, $N = 7$ for $\lambda = 5$) and that mean power consumption significantly drops with proper threshold selection. Simulations showed that incorporating the timer T in the Min(N,T) policy effectively reduces delay for low traffic scenarios. The results demonstrated a clear trade-off between delay and energy efficiency, with optimal configurations achieving substantial energy savings without compromising data transmission reliability, especially in bursty or low-frequency environments.

- **Huang et al. (2019)** [29] proposed a life extending approach for WSNs using an N-policy $M/M/1/K$ queuing model with multiple priority queues (high, medium, low). The sensor node operates with a finite buffer (size K) and triggers transmission only when the queue reaches a threshold N (e.g., $N = 4$), reducing frequent state transitions to save energy. The model employs a preemptive priority mechanism: if a higher-priority packet arrives during the processing of a lower-priority one, the sensor immediately interrupts the current task to handle the urgent packet, ensuring minimal latency for critical data. Power metrics were defined for sleep (0.003 mW), idle (30 mW), and transmit (81 mW) states, with MATLAB simulations analyzing trade-offs between energy efficiency and delay. Results showed that high priority packets experienced 30–50% lower delays, while an optimal $N = 4$ minimized power consumption. The study demonstrated that this approach balances energy savings with responsiveness, though larger N values introduce latency for lower-priority traffic.

- **Ghosh and Unnikrishnan (2017)** [2] investigated a queue-based approach to reduce power consumption in WSNs using an N-policy M/M/1 queuing model. The system operates with two primary states: an idle state, where the sensor node waits until its

buffer accumulates N packets, and a busy state, where the node transmits all buffered packets in burst mode. Their method relies on threshold-based transmission triggering to reduce frequent radio state transitions and dynamically adjusts the optimal threshold N^* based on the traffic intensity. The study considers three power modes corresponding to different operational states: transmit mode (81mW), receive/ idle mode (30mW), and sleep mode (0.003mW). Using MATLAB simulations with arrival rates ranging from $\lambda = 1.0$ to 5.0 and a fixed service rate of $\mu = 10$, they demonstrated significant energy savings of approximately 44–58% compared to systems without threshold-based control. The optimal thresholds varied with traffic conditions, ranging from $N = 4$ for $\lambda = 1.0$ to $N = 8$ for $\lambda = 5.0$. Their approach also effectively mitigated the energy-hole problem near sink nodes. The close match between analytical results and simulation outcomes confirmed the practical applicability of the proposed N-policy model in energy-constrained WSN deployments.

- **Boutoumi and Gharbi (2017)** [30] in their study *"An Energy Saving and Latency Delay Efficiency Scheme for Wireless Sensor Networks Based on GSPNs"* proposed a novel approach to optimize the performance of WSNs. Their model introduced two operational modes for sensor nodes: an N-policy mode, where nodes wake only when the packet queue reaches a predefined threshold N, and a hybrid policy mode, which combines this threshold mechanism with random wakeups to mitigate latency issues. The system was modeled using GSPNs, allowing for precise representation of complex node behavior. The authors assumed a finite buffer capacity of 10 packets and incorporated realistic traffic sources and three power states, each with different energy consumption levels. Through simulations conducted with the GreatSPN tool, the hybrid policy showed a significant reduction in packet delays, achieving 30–50% lower latency compared to the pure N-policy mode. Optimal thresholds were identified in the range of $N = 3$ to 8, depending on the traffic intensity, and the model demonstrated an effective balance between energy efficiency and responsiveness. The study offered both theoretical modeling and practical guidance, highlighting the benefits of GSPN-based analysis for implementing adaptive wakeup strategies in delay-sensitive WSN applications.

Table 1.1: Summary of Related Works on N-Policy and Queue Based Models in WSNs

References	Proposed Model	Methodology	Performance Indices
[26]	M/M/1 Queueing with N-Policy	Analytical model	Energy consumption, average delay
[5]	M/M/1 Queueing	Analytical model, simulation	Total energy consumption, average queue length
[27]	M/M/1/C Queueing with N-policy	Analytical model, simulation	Average power consumption, average delay, timely reliability
[28]	M/M/1 Queueing system	Analytical model	Average energy consumption
[3]	M/M/1 Queueing with N-policy min(N,T)	CTMC, analytical model	Power consumption, average delay
[29]	M/M/1/K Queueing with priorities N-policy	Analytical model, MATLAB simulation	Average delay, power consumption
[2]	M/M/1 Queueing	Analytical model, simulation	Average power consumption
[30]	M/M/1/K Hybrid N-policy with GSPN	GSPN model, simulation	Energy consumption, waiting delay

1.7. Conclusion

This chapter outlined the theoretical tools needed to analyze energy-efficient vacations policies in WSNs, covering WSN architecture, the energy-delay tradeoff, and relevant queueing models such as those with threshhold vacations. We also introduced CTMCs for modeling systems, and discussed DES techniques for evaluating system performance. These foundations support the modeling and evaluation of an N-policy queueing system with batch service in the next chapter.

Chapter 2

Modeling of the Sensor Node System

2.1. Introduction

While reviewing prior queue-based strategies, we noted that many rely on classical N-policy models with simplified two state transitions and often assume unrealistic conditions such as infinite buffer capacity. Among the literature, two studies stood out: [5] proposed a hybrid N-policy model integrating batch service, though without finite-buffer constraints or detailed Markovian analysis also [27], who introduced a wait state between activity and sleep, allowing the system to delay idle transitions and potentially reduce radio activations. These works inspired us to investigate how both batch processing and wait-state might be adapted to more realistic sensor node policies.

This chapter builds on those questions. We explore whether batch service can enhance energy efficiency in constrained systems, and whether introducing a wait state could limit excessive wake ups by briefly postponing idle transitions. To address these hypotheses, we design a set of queuing models based on operational policies applied to a wireless sensor node. Before introducing models that incorporate batch processing, we first assess its feasibility under typical WSN limitations, as discussed in the following section.

2.2. Evaluating the Practical Feasibility of Batch Service in WSNs

In this work, we investigate the feasibility of employing batch service mechanisms in WSNs as a strategy for energy optimization and latency reduction. Batch service, defined as the simultaneous or sequential processing of multiple packets during a single service period, offers the potential to reduce operational overhead such as idle listening, frequent state

switching, and redundant channel access. To assess the practical implementability of batch service, we first explore concurrent transmission as a theoretical candidate. The study reveals that while concurrent transmission aligns conceptually with batch processing in Markovian models (i.e., transitions from state m to $m - B$), it is largely impractical in WSNs due to the limitations in multi-radio architectures. Challenges include task scheduling complexities, routing inefficiencies, high hardware costs, and the absence of compatible MAC protocols for true intra node concurrency. These findings are elaborated in Appendix **Investigation of Concurrent Transmission**.

Given these constraints, we shift focus toward more realistic sequential batch transmissions using existing MAC-layer capabilities. Specifically, IEEE 802.15.4's Time Slotted Channel Hopping (TSCH) and Deterministic and Synchronous Multichannel Extension (DSME) modes both support tightly scheduled multi packet transmission through time-slot reservation mechanisms. Such features, including deterministic link assignment, synchronized channel hopping, and guaranteed time slots (GTS), enable batch like behavior without requiring concurrent transmission hardware. We provide a detailed analysis of these mechanisms and their alignment with batch service requirements in Appendix Sequential Transmission with TSCH and DSME.

In summary, while physical layer concurrency is not viable under current WSN constraints, batch service can be effectively approximated through MAC-level burst transmission mechanisms, particularly those standardized in IEEE 802.15.4. This realization supports the foundational modeling assumptions used in our queuing system and Markov chain-based energy analysis. For a complete technical discussion and literature-backed feasibility analysis, see Appendix A.

2.3. Queueing Model Foundations

This section outlines the fundamentals including queueing based abstraction, threshhold vacation policy, system parameters, and stochastic variables that serve as the basis for all subsequent proposed models.

2.3.1. System Abstraction

In our queueing model, the wireless sensor node system is abstracted using a classical M/M/1/K queueing structure. This abstraction assumes that packet arrivals follow a Poisson process with exponential interarrival times, service durations are exponentially distributed, the system includes a single server (the sensor node), and the buffer capacity is finite. Within

this framework, each data packet is treated as a customer in the queue, while the sensor node acts as the server responsible for processing these packets.

2.3.2. Threshold Vacation Policy

All the proposed models in this work adopt an exhaustive threshhold vacation policy which is the N-policy, where the server enters a vacation once the system becomes empty and remains idle until the number of packets in the system reaches a predefined threshold N. This threshold acts as the vacation termination condition, prompting service resumption only when the queue has accumulated enough traffic to justify activation.

2.3.3. Common System Parameters

All models discussed in this document are built upon the queueing abstraction outlined above and typically share a common set of system parameters that influence the node's behavior and overall performance. These parameters are presented here to establish a consistent analytical and modeling foundation. Packet arrivals at the sensor node are assumed to follow a Poisson process with rate λ, and service times are exponentially distributed. Two service rates are considered: μ_S for individual (single packet) processing and μ_B for batch processing. The batch service rate is defined as shown in equation 2.1 where $\alpha \in [1, 2]$ is a scaling parameter that regulates the intensity of batch service relative to single service. This is crucial, as unregulated batch throughput may exceed operational limits specified by IEEE standards for wireless sensor networks, and N is the threshold parameter.

$$\mu_B = \frac{\alpha \cdot \mu_S}{N} \tag{2.1}$$

In batch enabled models, the number of packets served in batch mode is denoted by B, which is computed as shown in equation 2.2, where $\beta \in [0, 1]$ is a proportionality constant ensuring that the batch size B scales with the threshold N.

$$B = \lceil \beta \cdot N \rceil \tag{2.2}$$

The node operates with a system capacity K. In certain configurations, a waiting period may precede service activation. when present, this delay is modeled as an exponentially distributed random variable with rate parameter θ. Finally, a maximum waiting threshold W is defined such that if the number of packets in the buffer reaches W during the wait state, the system aborts waiting and transitions back to service.

2.3.4. Stochastic State Variables

The sensor node models can be represented as a two dimensional CTMC. Each state is defined by a pair of stochastic processes, $X(t)$ and $Y(t)$, for $t \geq 0$. In this context, $X(t)$ denotes the number of packets currently stored in the buffer, while $Y(t)$ indicates the current operational mode of the node. The values of $Y(t)$ are defined as follows:

- $Y(t) = 0$ the node is in the *Idle State*,
- $Y(t) = 1$ the node is in the *Batch Service State*,
- $Y(t) = 2$ the node is in the *Single Service State*,
- $Y(t) = 3$: the node is in the *Wait State*.

Not all models make use of every operational state. The specific set of active states depends on the policy defined for each configuration and will be detailed accordingly in the respective model sections.

2.4. Basic Simulation Configuration

Before describing the specific characteristics of each proposed model, this section outlines the shared simulation framework and assumptions that are common to all configurations. All seven models are built upon the same underlying DES structure, differing only in how service policies are triggered and how the server behaves under different states.

2.4.1. Simulation Assumptions

In this simulation study, we use the Replication Method as previously described in Chapter 1. Each simulation run lasts for 1,000,000 units of time, where the unit is left intentionally abstract. To eliminate transient effects, a warmup period equal to 10,000 time units is discarded from the steady-state statistics for each simulation run.

2.4.2. Simulation Data Structures

The simulation is organized around four essential data structures: a priority queue that schedules upcoming events, and supporting structures that represent simulation events, input parameters, and collected results.

Priority queue: The priority queue detailed in structure 2.1, is responsible for managing the FEL. It ensures that events are processed in chronological order, always selecting the one with the earliest timestamp for execution.

Algorithm 2.1: Priority queue structure and operations

```
structure PriorityQueue:
    events : list<Event>;
    function push(event: Event)
        // Inserts a new event into the queue, maintaining the
           time order
    function pop() → Event
        // Removes and returns the event with the earliest time
```

Simulation parameters: This structure, shown in structure 2.2, contains all configurable inputs for one simulation run. These parameters determine traffic characteristics, system constraints, and behavioral thresholds such as batch size or activation policy.

Algorithm 2.2: Simulation parameters structure

```
structure SimulationParams:
    lumbda : float;
    mu_batch : float;
    mu_single : float;
    theta : float;
    N : integer;
    K : integer;
    B : integer;
    SIM_TIME : float;
    WARMUP_TIME : float;
    RD_ITERATION_NB : integer;
```

Event structure: The Event structure, demonstrated in structure 2.3, defines a discrete simulation event. Each event contains a scheduled time, a label indicating the event type (e.g., arrival or departure), and a flag indicating whether it belongs to a batch service.

Algorithm 2.3: Event data structure

```
structure Event:
    time : float;
    type : string;
    is_batch : boolean;
```

Simulation results: This structure, detailed in structure 2.4, accumulates the statistics generated during a simulation. These metrics include the proportion of time spent in each system state, throughput, average queue size, and the rate of packet loss or activation.

Algorithm 2.4: Simulation results structure

```
structure SimulationResults:
    p_idle : float;
    p_batch : float;
    p_single : float;
    p_wait : float;
    dropped_packets_rate : float;
    activation_rate : float;
    average_packet_nb : float;
    batch_service_throughput : float;
    single_service_throughput : float;
    sojourn_time : float;
```

2.4.3. Simulation clock and time-advancing mechanism

In our discrete event simulation, time progression is managed through a simulation clock, which advances by processing events in chronological order. At each iteration, the next scheduled event is retrieved using the priority queue, and the clock jumps directly to that event's timestamp.

2.4.4. Generating arrival, service, and timeout durations

To generate random durations for inter arrivals, service times, or waiting timeouts in the simulation, we use the exponential distribution via the inverse transform method. The time t is computed as illustrated in equation 2.3, where R is a uniform random variable between 0 and 1, and the *parameter* corresponds to the rate of the desired process (e.g., arrival rate λ, batch service rate μ_B, single service rate μ_S, or timeout rate θ).

$$t = \frac{\log(1 - R)}{-\text{parameter}} \tag{2.3}$$

2.4.5. Initialization procedure

The initialization phase prepares the system state and data structures before simulation begins. The server starts in the IDLE state, and the simulation clock is set to zero. The event list is initialized, and statistical accumulators are reset. Finally, the first packet arrival is scheduled as outlined in procedure 2.5.

Algorithm 2.5: Simulation initialization procedure

Procedure `initialize`(*simulation_params: SimulationParams*)

$current\_state \leftarrow IDLE$;
$current\_time \leftarrow 0.0$;
$queue\_size \leftarrow 0$;
$current\_time\_out \leftarrow 0.0$;
$total\_queue\_size\_time \leftarrow 0.0$;
$time\_idle \leftarrow 0.0$;
$time\_batch \leftarrow 0.0$;
$time\_single \leftarrow 0.0$;
$time\_wait \leftarrow 0.0$;
$batch\_service\_count \leftarrow 0$;
$single\_service\_count \leftarrow 0$;
$dropped\_packets\_count \leftarrow 0$;
$activation\_count \leftarrow 0$;
$arrived\_packets \leftarrow 0.0$;
$events \leftarrow$ `PriorityQueue()`;
push_arrival_event();

2.4.6. Simulation events

In our simulation, three primary events govern the system's behavior (arrival, departure and wait timeout). Each of these events triggers a corresponding handler procedure that updates the simulation state and schedules future events as necessary. Although the internal logic of these handlers varies depending on the specific policy being modeled, we observed that some operations are consistent across all configurations. To avoid redundancy in the model-specific sections, we define here the common logic shared among all models, in algorithm 2.6. The procedure for handling the wait timeout event is identical across all models that include it.

In addition, the arrival event handler, the single departure handler, and the batch departure handler each contain a common section of logic shared across all models.

Algorithm 2.6: Unified event-handling procedures

Procedure

- **if** $queue\_size < K$ **then**
  - $queue\_size \leftarrow queue\_size + 1$;
  - $arrived\_packet \leftarrow arrived\_packet + 1$;
- **else**
  - $dropped\_packets\_count \leftarrow dropped\_packets\_count + 1$;
- push_arrival_event();
- **if** $current\_state = IDLE$ ***and*** $queue\_size \geq N$ **then**
  - $activation\_count \leftarrow activation\_count + 1$;

Procedure `handle_common_batch_departure_behavior(`*B: integer*`)`

- $queue\_size \leftarrow queue\_size - B$;
- $batch\_service\_count \leftarrow batch\_service\_count + B$;

Procedure `handle_common_single_departure_behavior()`

- $queue\_size \leftarrow queue\_size - 1$;
- $single\_service\_count \leftarrow single\_service\_count + 1$;

Procedure `handle_wait_timeout_event()`

- **if** $current\_state \neq WAIT$ ***or*** $current\_time\_out > event.time$ ***or*** $current\_state = BATCH$ **then**
  - **continue**;
- **if** $queue\_size > 0$ **then**
  - $current\_state \leftarrow SINGLE$;
  - events.push({current_time + single_dist(generator), “departure”, false});
- **else**
  - $current\_state \leftarrow IDLE$;

2.4.7. Main program

The main program manages the simulation execution across different values of a chosen parameter. For each parameter value, it runs multiple simulation trials, aggregates statistics, and stores the averaged outcomes, see function in 2.7. Each simulation progresses by repeat-

edly fetching the next event, advancing the simulation clock, and invoking the corresponding event handler. After all replications are completed, the results are written to a file.

Algorithm 2.7: Main program function

```
function MainProgram()
    foreach target_value in parameter_range do
        simulation_params ← set_simulation_params(target_settings, target_value);
        result_sum ← initialize_result_sum();
        for i ← 1 to simulation_params.RD_ITERATION_NB do
            simulation_result ← run_simulation(simulation_params);
            sum_results(result_sum, simulation_result);
        averaged_results ← compute_average_results(result_sum,
         simulation_params.RD_ITERATION_NB);
        write_results_to_output_file(simulation_params, averaged_results);
```

Algorithm 2.8: Simulation execution function

```
function run_simulation(simulation_params: SimulationParams) → SimulationResults
    initialize(simulation_params);
    while current_time < simulation_params.SIM_TIME and not events.is_empty() do
        event ← events.pop();
        delta ← event.time − current_time;
        if current_time ≥ simulation_params.WARMUP_TIME then
            update_state_time(current_state, delta);
            total_queue_size_time ← total_queue_size_time + (queue_size × delta);
        current_time ← event.time;
        switch event.type do
            case "arrival" do
                handle_arrival_event();
            case "departure" do
                if event.is_batch then
                    handle_batch_departure_event();
                else
                    handle_single_departure_event();
            case "wait_timeout" do
                handle_wait_timeout_event();
    simulation_results ← compute_simulation_results();
    return simulation_results;
```

2.5. Proposed Queuing Models

This section introduces a structured classification of the proposed queuing models based on their primary service policies. The models are grouped into three distinct categories: Single Packet Service, One Time Batch Service, and Multiple Times Batch Service. Each category encompasses variants that explore different service dynamics under a unified activation threshold. Every model is described through a dedicated model description, a CTMC representation based on the general stochastic formulation established in section (2.3.4.) and a simulation event function tailored to the model's behavior. These simulation-specific procedures are provided here, complementing the core simulation architecture introduced earlier in section (2.4.6.).

2.5.1. Single-Packet Service Models

This section introduces vacations models that employ only single packet service, providing ground for comparison with more advanced models.

a. Model M0 (Baseline single service)

Model description: The Baseline Single Service model features a straightforward operational logic with only two distinct states: Idle and Single Service State. Initially, the sensor node resides in the idle state, consuming minimal energy while monitoring for incoming packets. Once the number of queued packets reaches the predefined threshold N, the system transitions to the single service state, where it begins processing packets individually. Unlike some other more complex models presented in this work, Model M0 does not employ batch processing. Instead, it operates under a simple continuous single service policy, in which packets are served one by one. During the single service state, the system continues to accept new packet arrivals until the buffer reaches its maximum capacity K. When the buffer is emptied, indicating that all queued packets have been processed, the node returns to the idle state, where it waits for the queue to refill to the activation threshold. This minimal two-state logic makes Model M0 the simplest among all models considered, providing a baseline for evaluating the performance impact of more sophisticated policies. Figure 2.1 outlines the operational process of Model M0, illustrating how the sensor node transitions between idle and single service states based on buffer occupancy.

CTMC modeling: Figure 2.2 presents the complete CTMC representation for this model, illustrating all permissible transitions between operational states. Algorithm 2.9 outlines the procedure used to construct the infinitesimal generator matrix, based on the $K + N$ states defined in the CTMC.

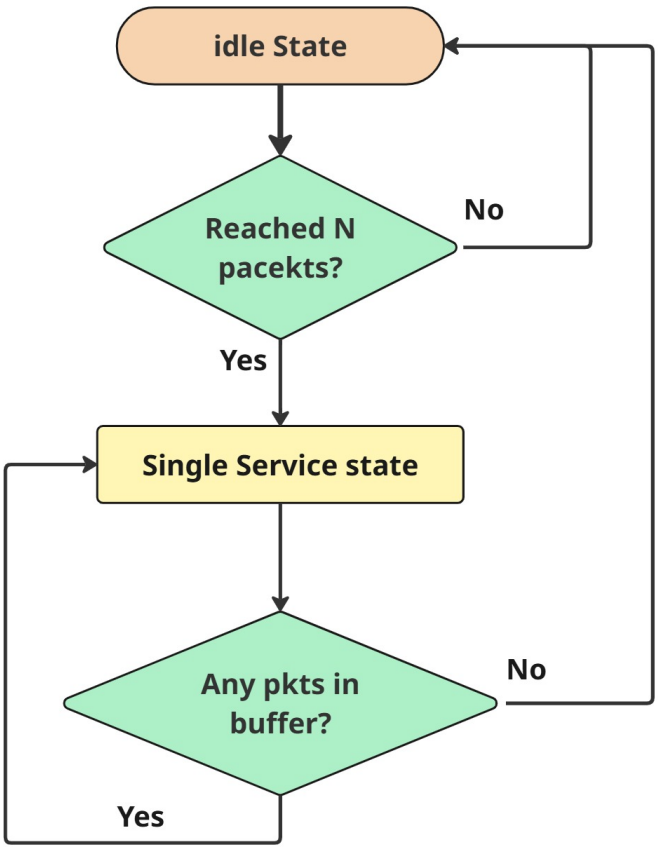

Figure 2.1: State transition flowchart of the sensor node for Model M0 (Baseline single service)

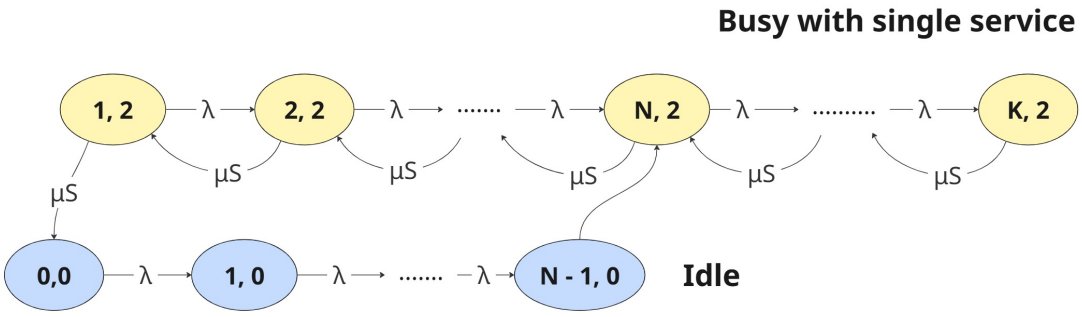

Figure 2.2: CTMC for model M0 (Baseline single service)

Algorithm 2.9: Algorithm for constructing the infinitesimal generator matrix (Model M0)

Data: K, N, μ_S, λ
Result: Q
Initialize Q with zeros;
for $i \leftarrow 0$ **to** $N - 2$ **do**
    $Q[i, i+1] \leftarrow \lambda$;
$Q[N-1, 2 \times N - 1] \leftarrow \lambda$;
for $i \leftarrow N$ **to** $K + N - 2$ **do**
    $Q[i, i+1] \leftarrow \lambda$;
    $Q[i+1, i] \leftarrow \mu_S$;
$Q[N, 0] \leftarrow \mu_S$;
for $i \leftarrow 0$ **to** $K + N - 1$ **do**
    $Q[i, i] \leftarrow -\text{sum\_row}(i)$;
return Q

Simulation event handling procedures: The event-handling procedures used in the simulation for the current model are detailed in Algorithm 2.10.

Algorithm 2.10: Event-handling procedures (Model M0)

Procedure `handle_arrival_event()`
    handle_common_arrival_behavior();
    if $current\_state = IDLE$ ***and*** $queue\_size \geq N$ **then**
        $current\_state \leftarrow SINGLE$;
        push_single_departure_event();

Procedure `handle_single_departure_event()`
    handle_common_single_departure_behavior();
    if $queue\_size > 0$ **then**
        $current\_state \leftarrow SINGLE$;
        push_single_departure_event();
    else
        $current\_state \leftarrow IDLE$;

b. Model M1 (Baseline single service with service hold)

Model description: The system begins in the idle state, where the sensor node accumulates incoming packets without initiating any processing. Once a sufficient number of packets have been collected which is equal to the threshold, the node transitions to the single service state. In the single service state, the node processes packets individually. During this phase, it continues to accept new arrivals, provided the buffer is not yet full. The system remains in this state until all queued packets have been served. Upon emptying the buffer, the node does not return directly to the idle state. Instead, it enters a temporary waiting state, during which it passively monitors for incoming packets while a countdown timer is active. Packet arrivals continue to be accepted during this period. The waiting state concludes under one of two conditions. If the buffer accumulates W packets before the timer expires, where $0 < W \leq K$. the node immediately returns to the single service state. Otherwise, if the timer elapses first, the node checks the buffer: if it remains empty, the system transitions to the idle state; if it contains any packets, the node resumes individual packet processing by returning to the single service state. Figure 2.3 presents a flowchart depicting the logical operation of Model M1, offering a visual overview of its internal behavior and the decision rules that guide state transitions according to buffer occupancy and service progression.

CTMC modeling: The full CTMC representation of this model is shown in Figure 2.4, adhering to the previously described modeling approach and encompassing all state transitions unique to this configuration. The infinitesimal generator matrix for this model is constructed using the procedure outlined in Algorithm 2.11, which considers a state space comprising $K + N + W$ distinct states.

Simulation event-handling procedures: Algorithm 2.12 presents the event-handling steps implemented for this simulation model.

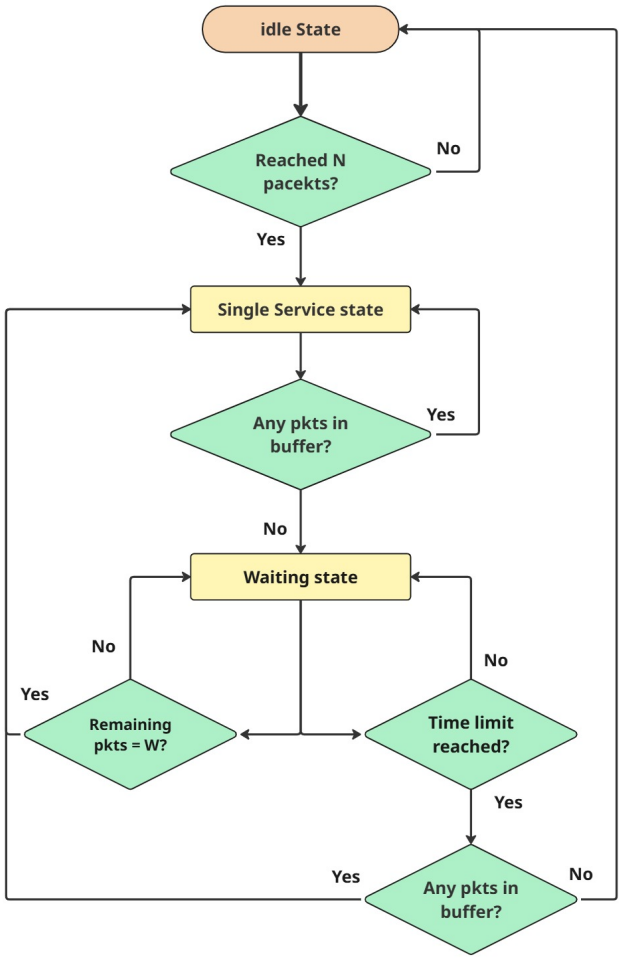

Figure 2.3: State transition flowchart of the sensor node for model M1 (Baseline single service with service hold)

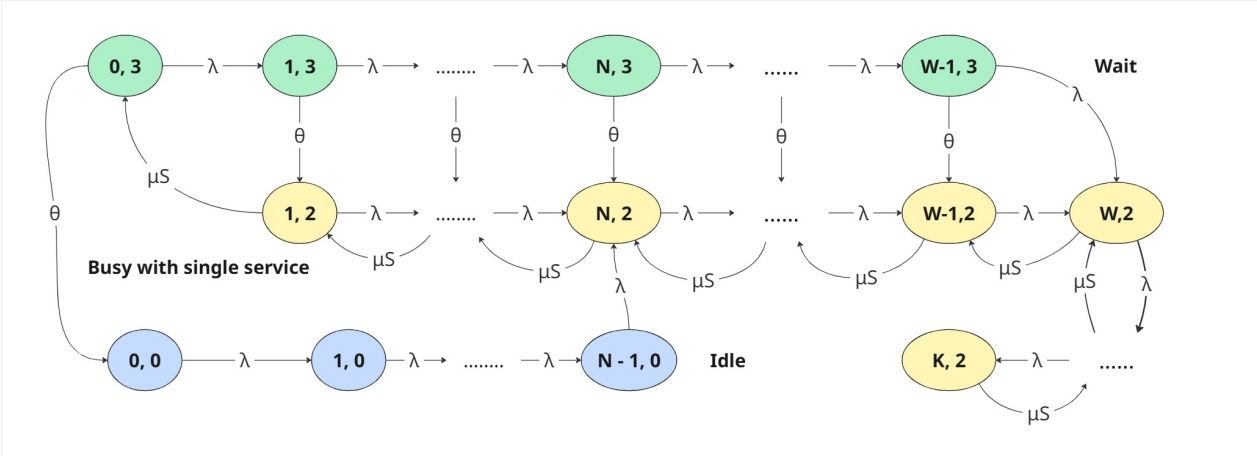

Figure 2.4: CTMC for Model M1 (Baseline single service with service hold)

Algorithm 2.11: Algorithm for constructing the infinitesimal generator matrix (Model M1)

Data: K, N, θ, μ_S, λ, W.

Result: Q.

Initialize Q with zeros;

for $i \leftarrow 0$ **to** $N-2$ **do**
  $Q[i, i+1] \leftarrow \lambda$;
$Q[N-1, 2\times N-1] \leftarrow \lambda$;
for $i \leftarrow N$ **to** $K+N-2$ **do**
  $Q[i, i+1] \leftarrow \lambda$;
  $Q[i+1, i] \leftarrow \mu_S$;
$Q[N, K+N] \leftarrow \mu_S$;
for $i \leftarrow K+N$ **to** $K+N+W-2$ **do**
  $Q[i, i+1] \leftarrow \lambda$;
  $Q[i+1, i-K] \leftarrow \theta$;
$Q[K+N+W-1, W+N-1] \leftarrow \lambda$;
$Q[K+N, 0] \leftarrow \theta$;
for $i \leftarrow 0$ **to** $K+N+W-1$ **do**
  $Q[i, i] \leftarrow -\text{sum\_row}(i)$;
return Q

Algorithm 2.12: Event-handling procedures (Model M1)

Procedure `handle_arrival_event()`
  handle_common_arrival_behavior();
  if $current\_state = IDLE$ ***and*** $queue\_size \geq N$ **then**
    $current\_state \leftarrow SINGLE$;
    push_single_departure_event();
  if $current\_state = WAIT$ ***and*** $queue\_size \geq W$ **then**
    $current\_state \leftarrow SINGLE$;
    $current\_time\_out \leftarrow -1$ push_single_departure_event();

Procedure `handle_single_departure_event()`
  handle_common_single_departure_behavior();
  if $queue\_size > 0$ **then**
    $current\_state \leftarrow SINGLE$;
    push_single_departure_event();
  else
    $current\_state \leftarrow WAIT$;
    $current\_time\_out \leftarrow push\_wait\_timeout\_event()$

2.5.2. One-Time Batch Service Models

This section presents models that execute a single batch service per busy period, after which the system transitions to another state based on the model description.

a. Model M2 (One-time batch service)

Model description: This model supports one-time batch service per cycle. The node alternates between three operational states: idle, batch service, and single service. It begins in the idle state, where incoming packets are accumulated without being processed. When the number of packets in the buffer reaches a predefined threshold N, the system transitions to the batch service state. At this stage, and before the batch service begins, additional packets may continue to arrive until the buffer reaches its maximum capacity K. During the batch service phase, exactly N packets are transmitted at the batch service rate μ_B. If the buffer held exactly N packets when the service started, the transmission depletes the buffer entirely, and the node returns to the idle state. Otherwise, if more than N packets were present, the remaining packets are processed individually in the subsequent single service state. In the single service state, packet arrivals resume as long as the buffer is not full. Packets are served one at a time at the single service rate μ_S. When the buffer becomes empty, the system returns to the idle state and remains there until the service threshold is reached again. The logical operation of Model M2 is illustrated in the flowchart shown in Figure 2.5. This diagram provides a visual summary of the internal behavior and decision rules that govern state transitions based on buffer occupancy and service progression.

CTMC modeling : The complete CTMC for this model is illustrated in Figure 2.6. It follows the modeling approach described earlier and captures the full range of state transitions specific to this configuration. This CTMC can be transformed into an infinitesimal generator matrix comprising $2K+1$ states, using the construction procedure outlined in Algorithm 2.13.

Simulation event-handling procedures: The logic for managing simulation events of the considered model is outlined in Algorithm 2.14.

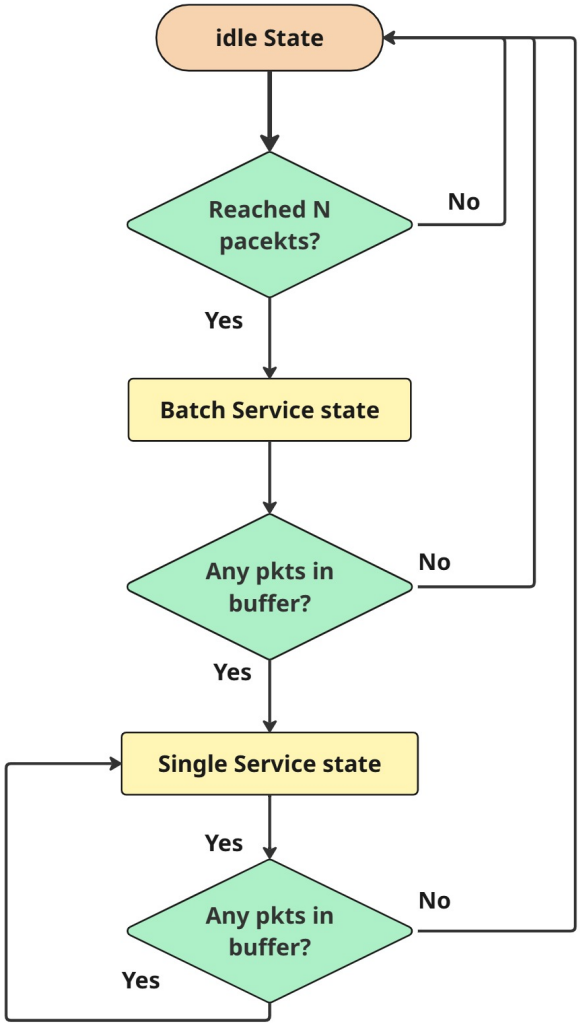

Figure 2.5: State transition flowchart of the sensor node for Model M2 (One-time batch service)

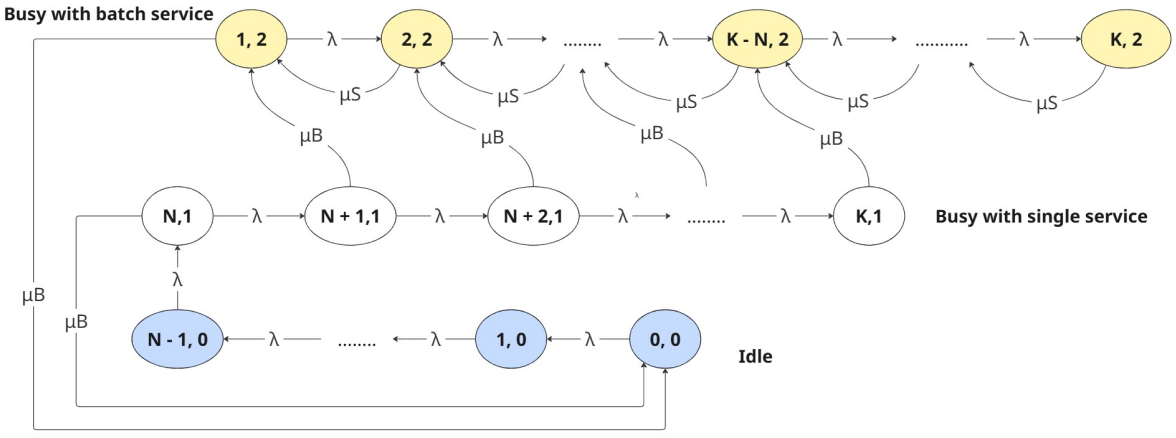

Figure 2.6: CTMC for Model M2 (One-time batch service)

Algorithm 2.13: Algorithm for constructing the infinitesimal generator matrix (Model M2)

Data: K, N, μ_B, μ_S, λ.

Result: Q.

Initialize Q with zeros;

for $i \leftarrow 0$ **to** $N - 1$ **do**
  $Q[i, i+1] \leftarrow \lambda$;

for $i \leftarrow N$ **to** $K - 1$ **do**
  $Q[i, i+1] \leftarrow \lambda$;
  $Q[i+1, i+1+(K-N)] \leftarrow \mu_B$;

$Q[N, 0] \leftarrow \mu_B$;

for $i \leftarrow K+1$ **to** $2K - 1$ **do**
  $Q[i, i+1] \leftarrow \lambda$;
  $Q[i+1, i] \leftarrow \mu_S$;

$Q[K+1, 0] \leftarrow \mu_S$;

for $i \leftarrow 0$ **to** $2 \times K$ **do**
  $Q[i, i] \leftarrow -\text{sum\_row}(i)$;

return Q

Algorithm 2.14: Event-handling procedures (Model M2)

Procedure `handle_arrival_event()`

```
handle_common_arrival_behavior();
if current_state = IDLE and queue_size ≥ N then
    current_state ← BATCH;
    push_batch_departure_event();
```

Procedure `handle_batch_departure_event()`

```
handle_common_batch_departure_behavior(N);
if queue_size > 0 then
    current_state ← SINGLE;
    push_single_departure_event();
else
    current_state ← IDLE;
```

Procedure `handle_single_departure_event()`

```
handle_common_single_departure_behavior();
if queue_size > 0 then
    current_state ← SINGLE;
    push_single_departure_event();
else
    current_state ← IDLE;
```

b. Model M3 (One-time batch service with service hold)

Model description: This model extends the behavior of Model M1 by integrating a limited waiting time following the single service phase. The system initiates in the idle state, where packets are accumulated until the number of buffered packets reaches the threshold N. Upon reaching this threshold, the node transitions into the batch service state and processes exactly N packets in a single batch. After the batch service operation, the system evaluates the buffer occupancy. If additional packets remain in the buffer, the node enters the single service state; otherwise, it reverts to the idle state. In the single service state, the node processes packets one at a time, continuing to accept new arrivals as long as the buffer is not full. This phase continues until the buffer becomes empty. At that point, instead of transitioning immediately to the idle state, the node enters a temporary waiting state. The waiting state concludes under one of two conditions. If the buffer accumulates W packets before the timer expires, where $0 < W \leq K$. the node immediately returns to the single service state. Otherwise, if the timer elapses first, the node checks the buffer: if it remains empty, the system transitions to the idle state; if it contains any packets, the node resumes individual packet processing by returning to the single service state. Figure 2.7 illustrates the flow-based representation of Model M3, capturing the state transition logic defined by its queuing policy

CTMC modeling : Figure 2.8 displays the complete CTMC for this model, following the previously outlined modeling methode and capturing all state transitions specific to this configuration. The corresponding CTMC can be converted into an infinitesimal generator matrix with $2\times K+W+1$ states, following the construction process detailed in Algorithm 2.15.

Simulation event-handling procedures: For this model, the event-handling functions are described in Algorithm 2.16.

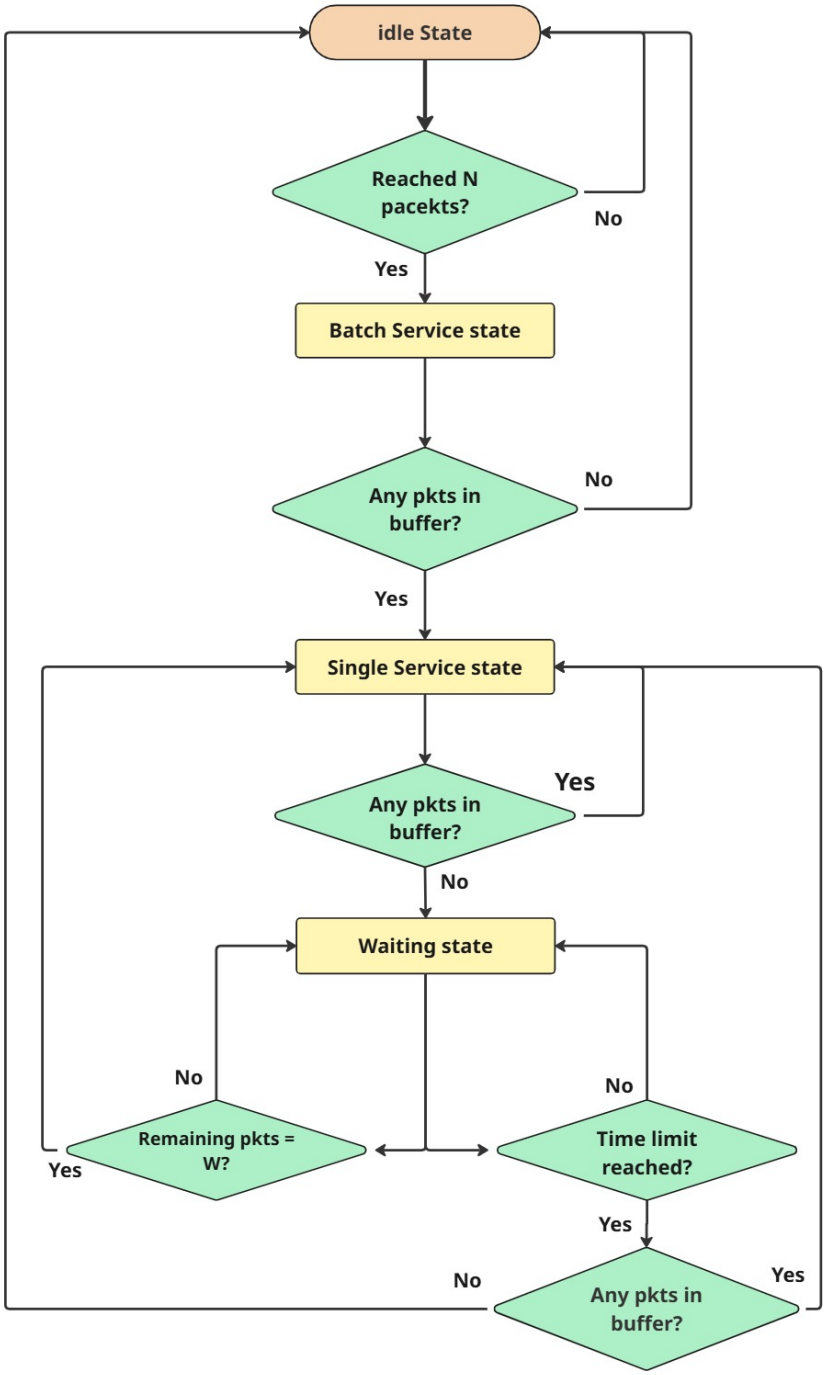

Figure 2.7: State transition flowchart of the sensor node for model M3 (One-time batch service with service hold)

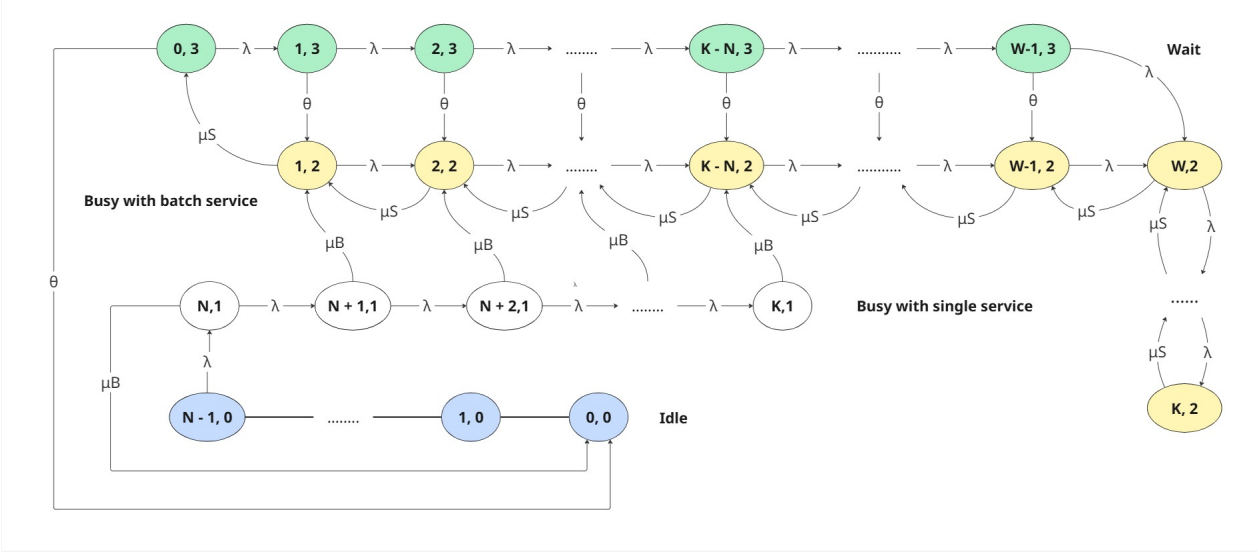

Figure 2.8: CTMC for model M3 (One-time batch service with service hold)

Algorithm 2.15: Algorithm for constructing the infinitesimal generator matrix (Model M3)

Data: K, N, μ_B, μ_S, λ, θ, W.

Result: Q.

Initialize Q with zeros;

for $i \leftarrow 0$ **to** $N-1$ **do**

  $Q[i, i+1] \leftarrow \lambda$;

for $i \leftarrow N$ **to** $K-1$ **do**

  $Q[i, i+1] \leftarrow \lambda$;

  $Q[i+1, i+1+(K-N)] \leftarrow \mu_B$;

$Q[N, 0] \leftarrow \mu_B$;

for $i \leftarrow K+1$ **to** $2K-1$ **do**

  $Q[i, i+1] \leftarrow \lambda$;

  $Q[i+1, i] \leftarrow \mu_S$;

$Q[K+1, 2\times K+1] \leftarrow \mu_S$;

for $i \leftarrow 2\times K+1$ **to** $2\times K+W-1$ **do**

  $Q[i, i+1] \leftarrow \lambda$;

  $Q[i+1, i-K] \leftarrow \theta$;

$Q[2\times K+W, K+W] \leftarrow \lambda$;

$Q[2\times K+1, 0] \leftarrow \theta$;

for $i \leftarrow 0$ **to** $2\times K+W$ **do**

  $Q[i, i] \leftarrow -\text{sum\_row}(i)$;

return Q

Simulation Event-Handling Procedures

Algorithm 2.16: Event-handling procedures (Model M3)

Procedure `handle_arrival_event()`

```
handle_common_arrival_behavior();
if current_state = IDLE and queue_size ≥ N then
    current_state ← BATCH;
    push_batch_departure_event();
if current_state = WAIT and queue_size ≥ W then
    current_state ← SINGLE;
    current_time_out ← −1;
    push_single_departure_event();
```

Procedure `handle_batch_departure_event()`

```
handle_common_batch_departure_behavior(N);
if queue_size > 0 then
    current_state ← SINGLE;
    push_single_departure_event();
else
    current_state ← IDLE;
```

Procedure `handle_single_departure_event()`

```
handle_common_single_departure_behavior();
if queue_size > 0 then
    current_state ← SINGLE;
    push_single_departure_event();
else
    current_state ← WAIT;
    current_time_out ← push_wait_timeout_event()
```

2.5.3. Multiple Batch Service Models

This section presents models that implement multiple-times batch service, where the sensor node may execute successive batch transmissions as long as threshold conditions are met.

a. Model M4 (Multiple-times batch service)

Model description: In this model, the sensor node operates under a multiple batch service policy. The system begins in the idle state, where it accumulates packets in the buffer. Once the number of queued packets reaches the predefined threshold N, the system transitions to the batch service state and prepares to serve a batch of B packets, where $1 < B \leq N$. Before the actual batch transmission begins, additional packets may continue to arrive until the buffer reaches its capacity K. During batch service, exactly B packets are transmitted. After each batch, the system reassesses the buffer content: if the number of remaining packets is greater than or equal to N, another batch service is immediately initiated; if the number of remaining packets is strictly less than N but not zero, the system transitions to the single service state; if the buffer becomes empty, the system returns directly to the idle state. In the single service state, packets are processed one by one, and new arrivals are still allowed as long as the buffer size remains below the threshold. If the buffer size again reaches the threshold during this phase, the system returns to the batch service state and resumes batch processing. The node returns to the idle state only when the buffer is completely empty. A flow oriented diagram of Model M4 is presented in Figure 2.9, summarizing the transition logic between node states under its queuing policy.

CTMC modeling : Figure 2.10 and Figure 2.11 give the CTMC representations corresponding to this model. Figure 2.10 presents the CTMC when the batch size B is fixed and equal to the threshold N (i.e., $B = N$), in which case $T = \left\lfloor \frac{K}{N} \right\rfloor$. Figure 2.11 corresponds to the case where the batch size satisfies $1 < B < N$, and in this case $T = \left\lfloor \frac{K-N+1}{B} \right\rfloor$. In both expressions, T denotes an Euclidean division. To generate the infinitesimal matrix for this model, we apply the construction procedure detailed in Algorithm 2.17, which accounts for a defined state space of K + N states.

Simulation event-handling procedures: The complete sequence of simulation event actions is provided in Algorithm 2.18.

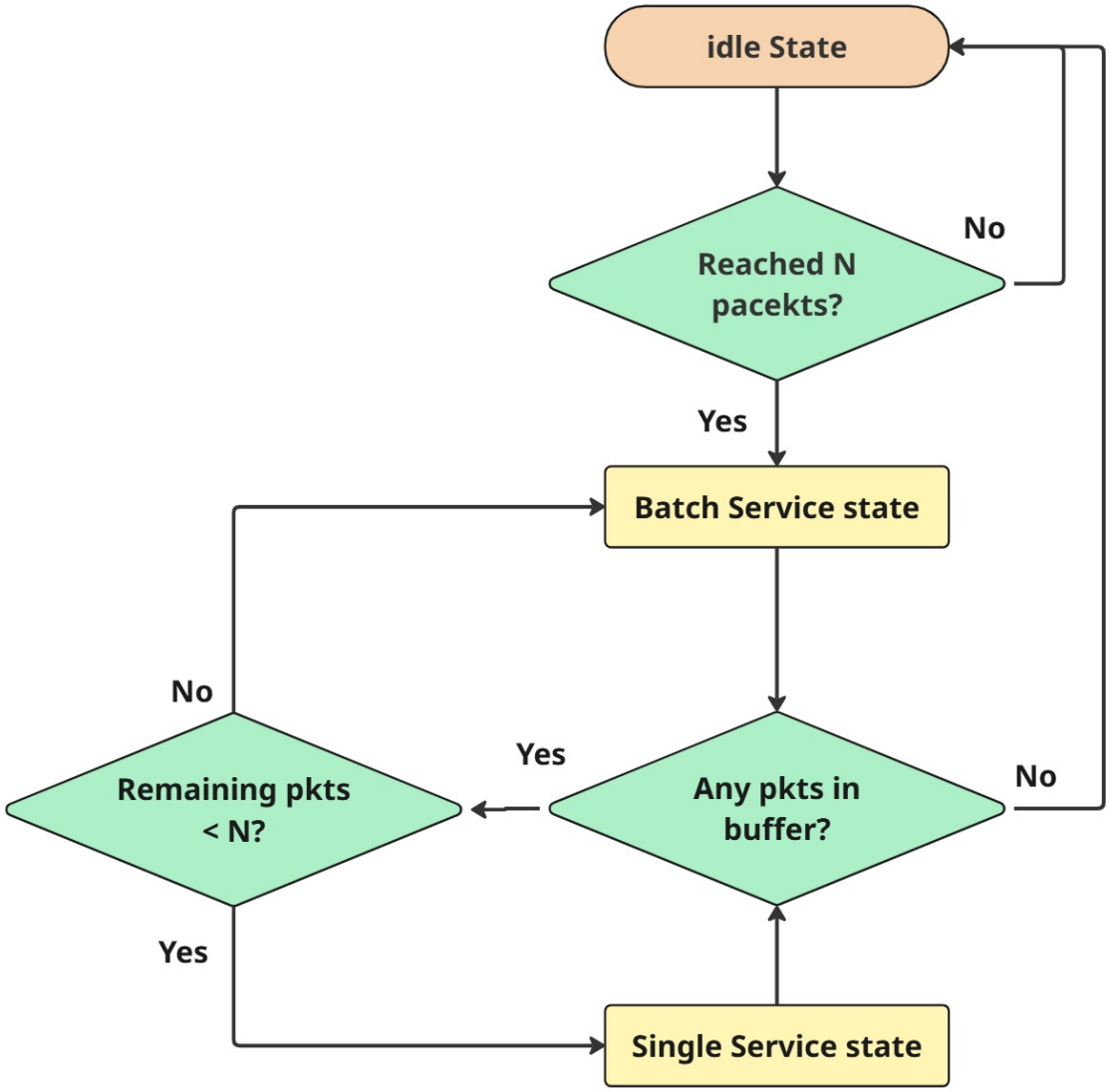

Figure 2.9: State transition flowchart of the sensor node for model M4 (Multiple Times Batch Service)

Algorithm 2.17: Algorithm for constructing the infinitesimal generator matrix (Model M4)

Data: K, N, μ_B, μ_S, λ, B, fst_ln_end.

Result: Q.

Initialize Q with zeros;

for $i \leftarrow 0$ **to** $K-1$ **do**
  $Q[i, i+1] \leftarrow \lambda$;

for $i \leftarrow K+1$ **to** $K+N-2$ **do**
  $Q[i, i+1] \leftarrow \lambda$;

$Q[K+N-1, N] \leftarrow \lambda$;

if $B = N$ **then**
  $Q[N, 0] \leftarrow \mu_B$;
  $fst\_ln\_end \leftarrow 2 \times N$;

else
  $Q[N, N+K-B] \leftarrow \mu_B$;
  $fst\_ln\_end \leftarrow N+B$;

for $i \leftarrow N+1$ **to** $\min(fst\_ln\_end-1, K)$ **do**
  $Q[i, i+(K-B)] \leftarrow \mu_B$;

if $K \geq fst\_ln\_end$ **then**
  for $i \leftarrow fst\_ln\_end$ **to** K **do**
    $Q[i, i-B] \leftarrow \mu_B$;

for $i \leftarrow K+1$ **to** $K+N-2$ **do**
  $Q[i+1, i] \leftarrow \mu_S$;

$Q[K+1, 0] \leftarrow \mu_S$;

for $i \leftarrow 0$ **to** $K+N-1$ **do**
  $Q[i, i] \leftarrow -\text{sum\_row}(i)$;

return Q

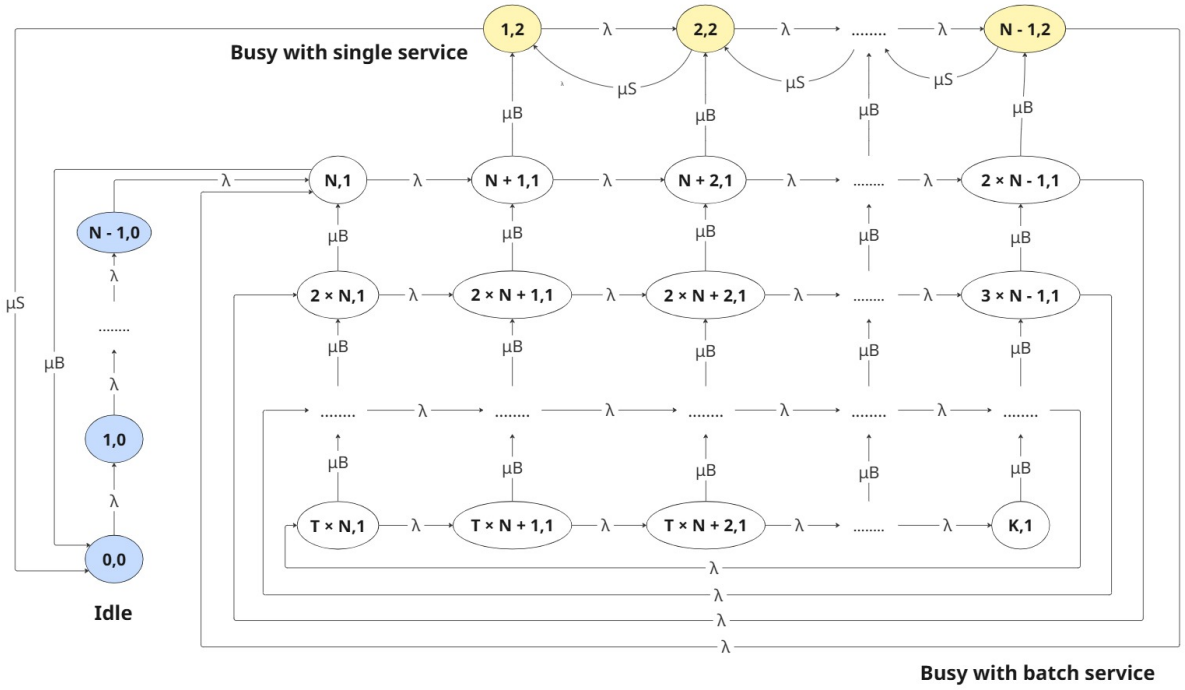

Figure 2.10: CTMC for model M4 (Multiple-times batch service, $B = N$)

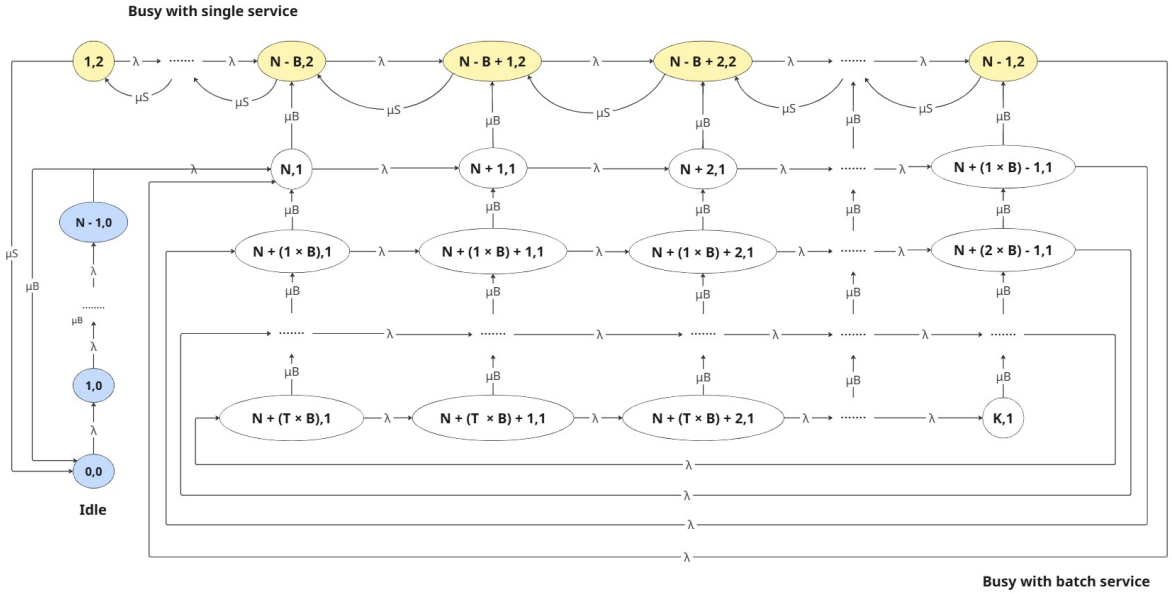

Figure 2.11: CTMC for Model M4 (Multiple-times batch service, $1 < B < N$)

Algorithm 2.18: Event-handling procedures (Model M4)

Procedure `handle_arrival_event()`

```
handle_common_arrival_behavior();
if (current_state = IDLE or current_state = SINGLE) and
  queue_size = N then
    current_state ← BATCH;
    push_batch_departure_event();
```

Procedure `handle_batch_departure_event()`

```
handle_common_batch_departure_behavior(B);
if queue_size ≥ N then
    current_state ← BATCH;
    push_batch_departure_event();
else if queue_size > 0 then
    current_state ← SINGLE;
    push_single_departure_event();
else
    current_state ← IDLE;
```

Procedure `handle_single_departure_event()`

```
handle_common_single_departure_behavior();
if current_state = BATCH then
    continue;
if queue_size > 0 and queue_size < N then
    current_state ← SINGLE;
    push_single_departure_event();
else
    current_state ← IDLE;
```

b. Model M5 (Multiple-times batch service with service hold)

Model Description: The system begins in the idle state, where it accumulates packets in the buffer. Once the number of queued packets reaches the predefined threshold N, the system transitions to the batch service state and prepares to serve a batch of B packets, where $1 < B \leq N$. Before the actual batch transmission begins, additional packets may continue to arrive until the buffer reaches its capacity K. During batch service, exactly B packets are transmitted. After each batch, the system reassesses the buffer content: if the number of remaining packets is greater than or equal to N, another batch service is immediately initiated; if the number of remaining packets is strictly less than N but not zero, the system transitions to the single service state; if the buffer becomes empty, the system returns directly to the idle state. In the single service state, packets are processed one by one, and new arrivals are still allowed as long as the buffer size remains below the threshold. If the buffer size again reaches the threshold during this phase, the system returns to the batch service state and resumes batch processing. The node transition to the wait state only when the buffer is completely empty. In the wait state, packets are still allowed to arrive but are not processed. Two outcomes govern the next transition. If the number of packets in the buffer reaches the threshold N before the timer expires, the system resumes batch processing by entering the batch service state. If the timer elapses before this threshold is met, the node evaluates the buffer. If it is still non-empty, it returns to the single service state. Otherwise, it transitions to the idle state. To complement the textual description, the flowchart in Figure 2.12 offers a clear procedural view of the decision logic underlying this Model.

CTMC modeling : Figures 2.13 and 2.14 present the CTMC models corresponding to this system. Figure 2.13 shows the case where the batch size B is equal to the threshold N, for which $T = \left\lfloor \frac{K}{N} \right\rfloor$. Figure 2.14 corresponds to the configuration where $1 < B < N$, yielding $T = \left\lfloor \frac{K-N+1}{B} \right\rfloor$. In both expressions, T is computed using Euclidean division. The CTMC structure is translated into an infinitesimal generator matrix comprising $2 \times N + K$ states, constructed using the steps outlined in Algorithm 2.19.

Simulation event-handling procedures: Algorithm 2.20 presents the event-handling steps implemented for this model.

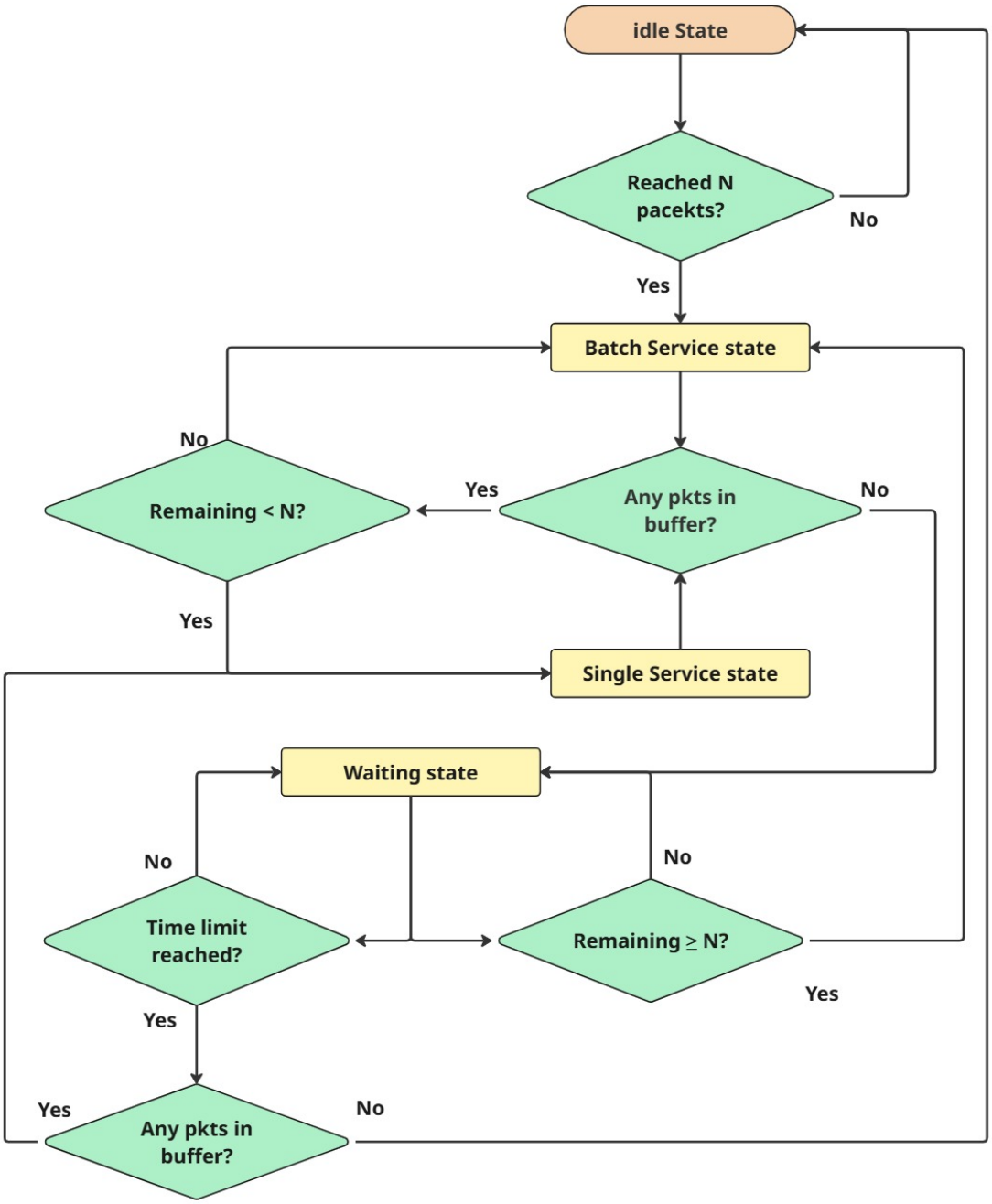

Figure 2.12: State transition flowchart of the sensor node for model M5 (Multiple-times batch service with service hold)

Algorithm 2.19: Algorithm for constructing the infinitesimal generator matrix (Model M5)

Data: K, N, μ_B, μ_S, θ, λ, B, fst_ln_end.

Result: Q.

Initialize Q with zeros;

for $i \leftarrow 0$ **to** $K-1$ **do**
  $Q[i, i+1] \leftarrow \lambda$;

for $i \leftarrow K+1$ **to** $K+N-2$ **do**
  $Q[i, i+1] \leftarrow \lambda$;
$Q[K+N-1, N] \leftarrow \lambda$;

for $i \leftarrow K+N$ **to** $2 \times N+K-2$ **do**
  $Q[i, i+1] \leftarrow \lambda$;
$Q[2 \times N+K-1, N] \leftarrow \lambda$;

if $B = N$ **then**
  $Q[N, 0] \leftarrow \mu_B$;
  $fst\_ln\_end \leftarrow 2 \times N$;

else
  $Q[N, N+K-B] \leftarrow \mu_B$;
  $fst\_ln\_end \leftarrow N+B$;

for $i \leftarrow N+1$ **to** $\min(fst\_ln\_end-1, K)$ **do**
  $Q[i, i+(K-B)] \leftarrow \mu_B$;

if $K \geq fst\_ln\_end$ **then**
  for $i \leftarrow fst\_ln\_end$ **to** K **do**
    $Q[i, i-B] \leftarrow \mu_B$;

for $i \leftarrow K+1$ **to** $K+N-2$ **do**
  $Q[i+1, i] \leftarrow \mu_S$;
$Q[K+1, 0] \leftarrow \mu_S$;

for $i \leftarrow K+N+1$ **to** $2 \times N+K$ **do**
  $Q[i, i-N] \leftarrow \theta$;
$Q[K+N, 0] \leftarrow \theta$;

for $i \leftarrow 0$ **to** $2 \times N+K-1$ **do**
  $Q[i, i] \leftarrow -\text{sum\_row}(i)$;

return Q

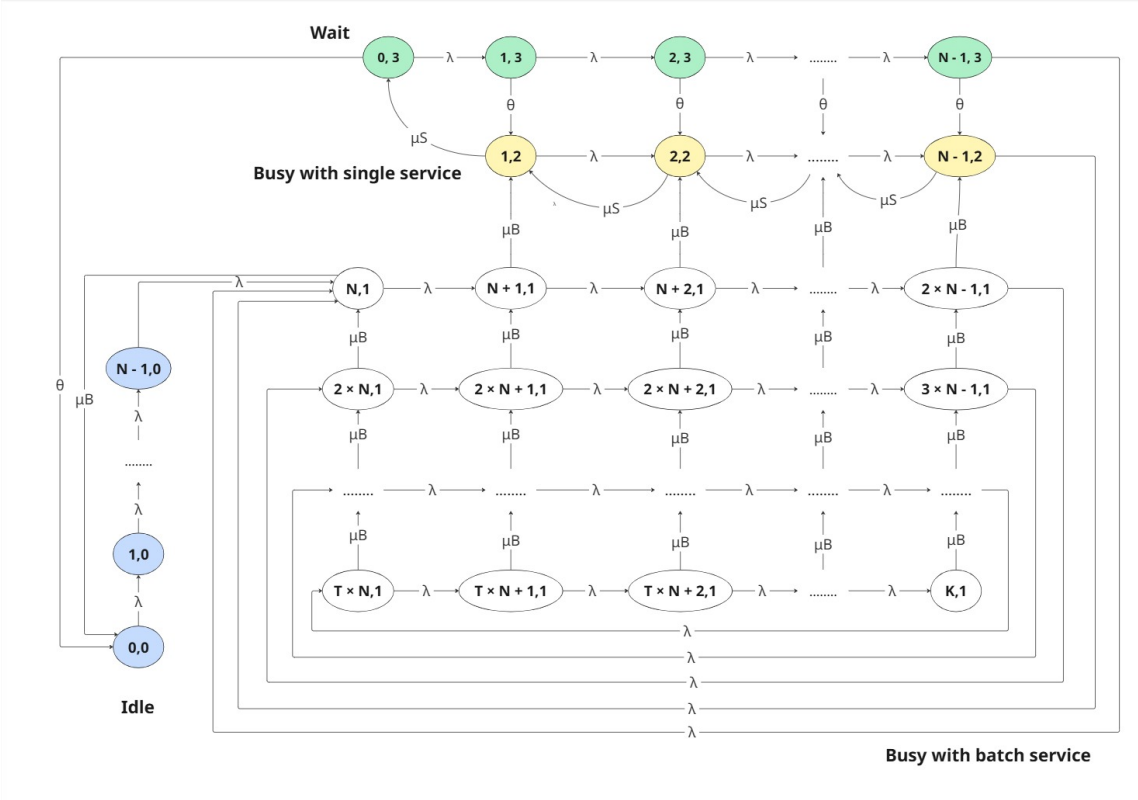

Figure 2.13: CTMC for model M5 (Multiple-times batch service with service hold, $B = N$)

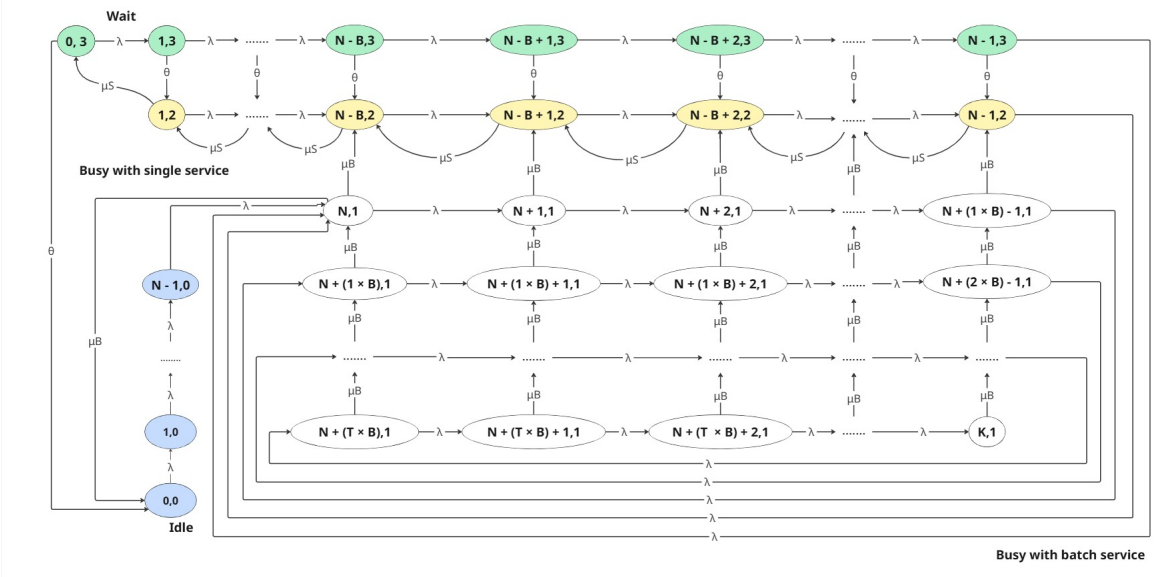

Figure 2.14: CTMC for model M5 (Multiple-times batch service with service hold, $1 < B < N$)

Algorithm 2.20: Event-Handling Procedures (Model M5)

```
Procedure handle_arrival_event()
    handle_common_arrival_behavior();
    if (current_state = IDLE or current_state = SINGLE or
     current_state = WAIT) and queue_size = N then
        if current_state = WAIT then
            current_time_out ← −1
        current_state ← BATCH;
        push_batch_departure_event();

Procedure handle_batch_departure_event()
    handle_common_batch_departure_behavior(B);
    if queue_size ≥ N then
        current_state ← BATCH;
        push_batch_departure_event();
    else if queue_size > 0 then
        current_state ← SINGLE;
        push_single_departure_event();
    else
        current_state ← IDLE;

Procedure handle_single_departure_event()
    handle_common_single_departure_behavior();
    if current_state = BATCH then
        continue;
    if queue_size > 0 and queue_size < N then
        current_state ← SINGLE;
        push_single_departure_event();
    else
        current_state ← WAIT;
        current_time_out ← push_wait_timeout_event()
```

2.6. Performance Metrics

In this section, we focus on how the performance metrics are calculated within our proposed modeling approach, using both simulation and CTMC analysis. We systematically present each performance metric along with its corresponding symbol and meaning, followed by the

equations used for its calculation. The complete list of performance metrics used in our study is presented in Table 2.1. The formulas used to derive these metrics from the CTMC-based analytical model are shown in Table 2.2, while their simulation-based computation is detailed in Table 2.3.

Table 2.1: Performance metrics used in the proposed models

Symbol	Definition
P_I	Probability of being in the Idle state.
P_B	Probability of being in the Batch service state.
P_S	Probability of being in the Single service state.
P_W	Probability of being in the Wait state.
P_L	Probability of dropping or losing packets (Probability of loss).
D_r	Number of packets dropped per unit of time (Drop rate).
$\overline{Q}_I$	Average number of packets in the system during the Idle state.
$\overline{Q}_S$	Average number of packets in the system during the Single service state.
$\overline{Q}_B$	Average number of packets in the system during the Batch service state.
$\overline{Q}_W$	Average number of packets in the system during the Wait state.
$\overline{Q}$	Average number of packets in the system.
A_r	Number of radio activations per unit of time (Activation rate).
T_{batch}	Number of packets served in batch mode per unit of time (Batch service throughput).
T_{single}	Number of packets served in single mode per unit of time (Single service throughput).
$\overline{\lambda}$	Number of packets effectively accepted into the system per unit of time (effective arrival rate).
T_{soj}	Sojourn time: average time a packet spends in the system, including time in the buffer and in service.
T_{sim}	Total simulation duration.
T_I	Cumulative time spent in the Idle state during the simulation.
T_B	Cumulative time spent in the Batch service state during the simulation.
	Continued on next page

Table 2.1 – continued from previous page

Symbol	Definition
T_S	Cumulative time spent in the Single service state during the simulation.
T_W	Cumulative time spent in the Wait state during the simulation.
Δt_i	Duration of the i^{th} interval used to accumulate weighted queue size.
q_i	Number of packets in the system during the i^{th} time interval.
Q_{int}	Time-weighted integral of the number of packets: $\sum_i q_i \cdot \Delta t_i$.
N_L	Total number of dropped (lost) packets during the simulation.
N_A	Total number of radio activations.
N_B	Number of packets served in batch mode.
N_S	Number of packets served in single mode.
N_{arr}	Number of packets arriving to the system during simulation.
EC_I	Energy consumption cost of the communication unit in the Idle state.
EC_B	Energy consumption cost of the communication unit in the Busy state (covers both batch and single service modes).
EC_W	Energy consumption cost of the communication unit in the Wait state.
EC_{act}	Energy consumption cost of radio activation.
EC_{drop}	Energy consumption cost of dropping a packet.
EC_H	Energy consumption cost of holding one packet in the buffer.
EC_{txB}	Energy cost of transmitting one packet in batch mode.
EC_{txS}	Energy cost of transmitting one packet in single mode.
E_{state}	Total energy consumed by the system due to remaining in various operational states.
E_{total}	Total energy consumption, including state energy, energy due to packet drops, radio activations, buffer holding, and transmissions in both batch and single modes.

We compute stationary probability vector $\pi = [\pi_1, \pi_2, \ldots, \pi_n]$ as shown in equation 1.2.

Table 2.2: Performance metric equations derived from the CTMC analytical model

Performance Metric	M0	M1	M2	M3	M4	M5
P_I	$P_I = \sum_{i=0}^{N-1} \pi_{i,0}$					
P_B	–		$\sum_{i=N}^{K} \pi_{i,1}$			
P_S	$\sum_{i=1}^{K} \pi_{i,2}$				$\sum_{i=1}^{N-1} \pi_{i,2}$	
P_W	–	$\sum_{i=0}^{W-1} \pi_{i,3}$	–	$\sum_{i=0}^{W-1} \pi_{i,3}$	–	$\sum_{i=0}^{N-1} \pi_{i,3}$
P_L	$\sum_{j \in \mathcal{M}} \pi_{K,j} \quad ; \mathcal{M} \subseteq \{0,1,2,3\}$					
D_r	$P_L \cdot \lambda$					

(continued on next page)

(continued from previous page)

Performance Metric	**M0**	**M1**	**M2**	**M3**	**M4**	**M5**
$\overline{Q}_I$	$\sum_{i=0}^{N-1} i \cdot \pi_{i,0}$					
$\overline{Q}_B$	–		$\sum_{i=N}^{K} i \cdot \pi_{i,1}$			
$\overline{Q}_S$	$\sum_{i=1}^{K} i \cdot \pi_{i,2}$				$\sum_{i=1}^{N-1} i \cdot \pi_{i,2}$	
$\overline{Q}_W$	–	$\sum_{i=0}^{W-1} i \cdot \pi_{i,3}$	–	$\sum_{i=0}^{W-1} i \cdot \pi_{i,3}$	–	$\sum_{i=0}^{N-1} i \cdot \pi_{i,3}$
$\overline{Q}$	$\overline{Q}_I + \overline{Q}_S$	$\overline{Q}_I + \overline{Q}_S + \overline{Q}_W$	$\overline{Q}_I + \overline{Q}_S + \overline{Q}_B$	$\overline{Q}_I + \overline{Q}_S + \overline{Q}_B + \overline{Q}_W$	$\overline{Q}_I + \overline{Q}_S + \overline{Q}_B$	$\overline{Q}_I + \overline{Q}_S + \overline{Q}_B + \overline{Q}_W$
A_r	$\pi_{N-1,0} \cdot \lambda$					

(continued on next page)

(continued from previous page)

Performance Metric	**M0**	**M1**	**M2**	**M3**	**M4**	**M5**
T_{batch}	–	$P_B \cdot \mu_B$	–	$P_B \cdot \mu_B$	–	$P_B \cdot \mu_B$
T_{single}	$P_S \cdot \mu_S$					
$\overline{\lambda}$	$(1 - P_L) \cdot \lambda$					
T_{soj}	$\frac{\overline{Q}}{\overline{\lambda}}$					
$E_{\mathbf{state}}$	$P_I \cdot EC_I$ $+P_S \cdot EC_B$	$P_I \cdot EC_I$ $+P_S \cdot EC_B$ $+P_W \cdot EC_W$	$P_I \cdot EC_I$ $+P_S \cdot EC_B$ $+P_B \cdot EC_B$	$P_I \cdot EC_I$ $+P_S \cdot EC_B$ $+P_B \cdot EC_B$ $+P_W \cdot EC_W$	$P_I \cdot EC_I$ $+P_S \cdot EC_B$ $+P_B \cdot EC_B$	$P_I \cdot EC_I$ $+P_S \cdot EC_B$ $+P_B \cdot EC_B$ $+P_W \cdot EC_W$
$E_{\mathbf{total}}$	$E_{\text{state}} + D_r \cdot EC_{\text{drop}} + A_r \cdot EC_{\text{act}} + \overline{Q} \cdot EC_H + T_{batch} \cdot EC_{txB} + T_{single} \cdot EC_{txS}$					

Table 2.3: Performance metric equations obtained through DES

Performance Metric	**M0**	**M1**	**M2**	**M3**	**M4**	**M5**
P_I	$\frac{T_I}{T_{\text{sim}}}$					
P_B	–		$\frac{T_B}{T_{\text{sim}}}$			
P_S	$\frac{T_S}{T_{\text{sim}}}$					
P_W	–	$\frac{T_W}{T_{\text{sim}}}$	–	$\frac{T_W}{T_{\text{sim}}}$	–	$\frac{T_W}{T_{\text{sim}}}$
D_r	$\frac{N_L}{T_{\text{sim}}}$					
Q_{int}	$\sum_i q_i \cdot \Delta t_i$					
$\overline{Q}$	$\frac{Q_{\text{int}}}{T_{\text{sim}}}$					

(continued on next page)

(continued from previous page)

Performance Metric	**M0**	**M1**	**M2**	**M3**	**M4**	**M5**
A_r	$\frac{N_A}{T_{\text{sim}}}$					
T_{batch}	–		$\frac{N_B}{T_{\text{sim}}}$			
T_{single}	$\frac{N_S}{T_{\text{sim}}}$					
$\overline{\lambda}$	$\overline{\lambda} = \frac{N_{\text{arr}} - N_L}{T_{\text{sim}}}$					
T_{soj}	$T_{soj} = \frac{\overline{Q}}{\overline{\lambda}}$					
$E_{\textbf{state}}$	$P_I EC_I$ $+P_S EC_B$	$P_I EC_I$ $+P_S EC_B$ $+P_W EC_W$	$P_I EC_I$ $+P_S EC_B$ $+P_B EC_B$	$P_I EC_I$ $+P_S EC_B$ $+P_B EC_B$ $+P_W EC_W$	$P_I EC_I$ $+P_S EC_B$ $+P_B EC_B$	$P_I EC_I$ $+P_S EC_B$ $+P_B EC_B$ $+P_W EC_W$
$E_{\textbf{total}}$	$E_{\text{state}} + D_r \cdot EC_{\text{drop}} + A_r \cdot EC_{\text{act}} + \overline{Q} \cdot EC_H + T_{batch} \cdot EC_{txB} + T_{single} \cdot EC_{txS}$					

Conclusion

In this chapter, we developed a comprehensive modeling framework to represent the behavior of a wireless sensor node using a queueing system approach. Beginning with a practical evaluation of batch service feasibility in WSNs. Building on this foundation, we introduced multiple queuing models ranging from basic single service to more sophisticated batch models.For each model, we began with a brief description and provided a corresponding flowchart to visually represent the system behavior. We then defined the associated CTMC, and derived the infinitesimal generator matrix. This process enabled us to compute the steady-state probabilities and extract key performance metrics related to energy consumption and delay. We also defined the common system parameters, state variables, and simulation setup. In the next chapter, we use the obtained values to conduct experiments, compute the main performance indices, and evaluate the impact of various parameters on the energy consumption across the different proposed models.

Chapter 3

Implementation, Experimental Results And Discussions

3.1. Introduction

In this chapter, we evaluate the performance of the models proposed in the previous chapter using both analytical modeling based on CTMCs and simulation. Several experiments were conducted to study the impact of varying key parameters on a sensor node's performance. An application was developed to compute the performance metrics and facilitate the experimentation process. A screenshot of the application is included to illustrate its role. The results of the experiments are then presented and analyzed to compare the effectiveness of the different models.

3.2. Development Environment

As with any experimentation process, the development environment plays a crucial role in the implementation and testing of algorithms, as well as in the accurate interpretation of the obtained results.

3.2.1. Hardware

We used a laptop with the characteristics and specifications shown in Table 3.2:

Table 3.1: Specifications of the development environment

Brand	HP
RAM	64 GB
Processor	Intel Xeon 4210 (2,2 GHz, Intel Turbo Maxi, 13,75 Mo cache, 10 cœurs)
Operating System	Windows 11 64-bit

3.2.2. Software

The development of our algorithms was carried out entirely on a Windows operating system, using the following software tools:

- **Visual Studio Code**: a lightweight code editor with support for debugging, task execution, and version control.
- **Python**: a high-level, object-oriented programming language known for its simplicity and versatility, widely used for scientific computing and rapid prototyping.
- **C++**: used for implementing performance-critical parts of the application, especially in areas involving intensive algorithmic processing.

3.3. Implementation

The models developed in this study were implemented using Python and C++. The CTMC based mathematical formulation was integrated into the application to perform analytical evaluations and compute performance metrics such as energy consumption and packet delay.

To support user interaction and experimentation, we also developed a graphical user interface using the pyQt6 library in Python. The interface design and functionality are detailed in the following subsection.

3.3.1. Presentation of the Graphical Interfaces

The graphical interface provides a practical and intuitive environment for interacting with the system. Users can define input parameters for the CTMC model, trigger computations, and view the resulting performance metrics.

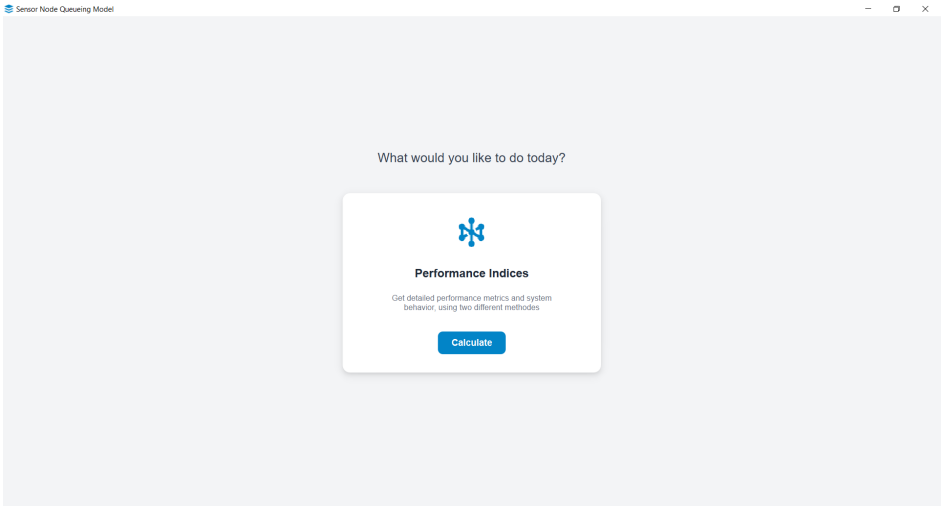

Figure 3.1: Main interface of the application.

Figure 3.1 shows the home screen of the application. From this interface, the user can access the main functionality through a single card labeled Performance Analysis. This option guides the user through setting up and running a simulation or configuring an analytical model. Depending on the selected approach, the application then computes performance metrics based on the defined parameters.

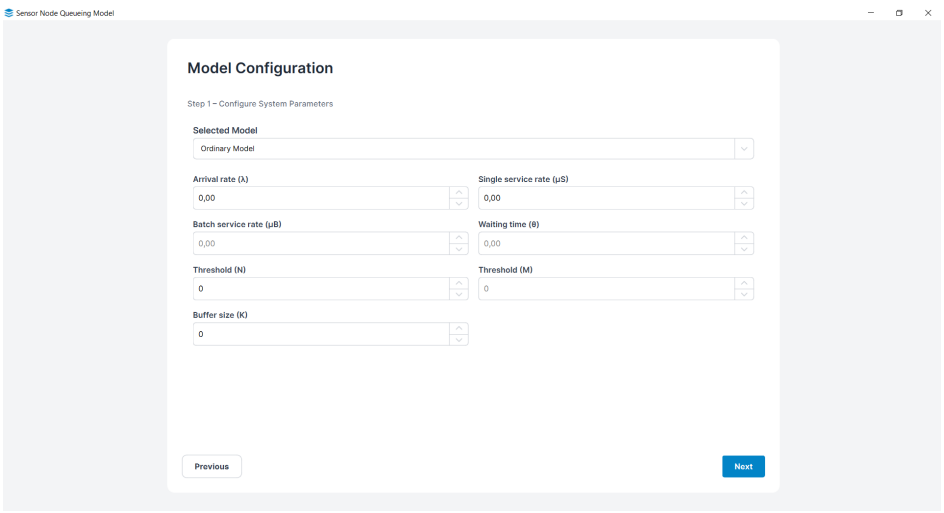

Figure 3.2: System parameter configuration screen.

As shown in Figure 3.2, the first step of the experiment configuration involves setting the queueing model parameters. These include the arrival rate (λ), service rates, system capacity, and threshold values that define how the sensor node operates. This section provides the essential inputs required for both simulation and CTMC modeling.

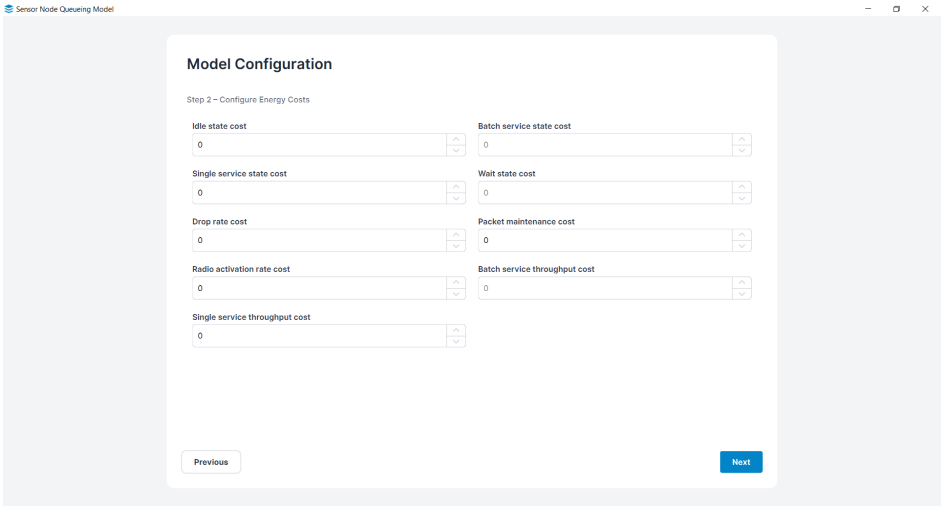

Figure 3.3: Energy cost configuration screen.

Figure 3.3 presents the second step in the configuration process, where users define the energy cost associated with different system states and operations. These values are necessary for calculating the total energy consumption during the performance evaluation phase.

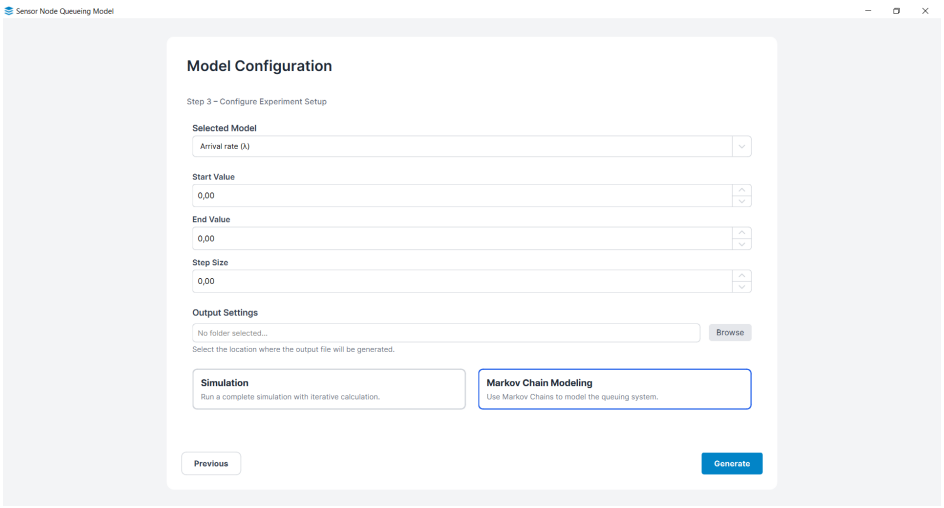

Figure 3.4: Experiment setup screen.

In the final configuration step, illustrated in Figure 3.4, users specify a range of values for the parameter to be tested by setting the start value, end value, and step size. Additionally, they can choose between running a full simulation or using a Markov Chain model for performance computation. The results are then generated and exported to the selected output location.

3.4. Experiments

To evaluate the effectiveness of the developed models and their ability to assess the performance of sensor nodes under various conditions, we conducted a series of experiments. These tests were designed to observe the impact of parameter variations.

The fixed energy cost values used in our experiments were derived and scaled from real hardware benchmarks reported in both datasheets and experimental studies. Specifically, we referred to the TelosB and MICAz mote platform datasheets [31, 32].

Additionally, we consulted two experimental benchmark studies [33, 34]. Based on these references, the abstract energy cost values were selected to be proportional to the actual energy draw of each state. The cost values used throughout all experiments are summarized in Table 3.2.

Table 3.2: Specifications of Energy Cost

EC_I	15
EC_B	240
EC_W	70
EC_{drop}	10
EC_H	3
EC_{act}	70
EC_{txB}	3
EC_{txS}	5

3.4.1. System Parameters

The following parameters were fixed and used consistently throughout all experiments to ensure a fair comparison between the different models. These values were selected based on preliminary experiments conducted to explore their influence on system behavior. We chose the parameter set that added meaningful insight to our evaluation and highlighted the strengths and limitations of each model. Due to the high complexity of the system and the large number of interacting parameters, it was not feasible to exhaustively test all possible combinations.

Table 3.3: Specifications of System Parameters

λ	6
μ_S	7
α	1.8
θ	0.5
K	50
W	15
β	0.8

3.4.2. Validation of Analytical Models Against Simulation

To assess the degree of consistency between the results of our simulation model and the analytical solution derived from the CTMC, we employ the Symmetric Mean Absolute Percentage Error (sMAPE). This metric, introduced by Armstrong [35] and further discussed by Hyndman and Koehler [36], is designed to quantify the relative difference between two sets of values, without assuming that either one is the true reference. Unlike standard percentage error metrics such as MAPE, sMAPE treats both the simulated and analytical results symmetrically by dividing the absolute error by the average of the two values. This prevents bias toward either value and makes it particularly good for situations where there is no absolute "ground truth". Additionally, sMAPE outputs are expressed as percentages, which allows for consistent interpretation across performance metrics that vary in scale and unit. Although sMAPE was originally developed for forecasting applications, its mathematical formulation is general-purpose and not restricted to time-series or predictive models. In this work, we adopt sMAPE as a model-agnostic error metric to evaluate the relative agreement between simulation and analytical results, independent of units or scales. As a result, sMAPE provides a robust and interpretable measure of relative difference, even when comparing metrics of different magnitudes or physical meanings. The formal definition of sMAPE is given in Equation 3.1.

$$\text{sMAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|x_i - y_i|}{(|x_i| + |y_i|)\,/2} \tag{3.1}$$

While there is no strict theoretical rule for interpreting sMAPE values, we use practical and intuitive ranges to evaluate the level of agreement between models. A sMAPE between 0% and 5% indicates excellent agreement, 5% to 10% suggests a good match, 10% to 20% reflects a moderate difference, and values above 20% indicate a significant mismatch. These ranges help to contextualize the relative difference between simulation and analytical results across performance metrics with different scales. The sMAPE values obtained for each model

and metric are summarized in Figure 3.5. The vast majority of these results indicate excellent agreement, with sMAPE values between 0% and 5%. Some metrics fall within the 5% to 10% range, indicating a good match. In particular, the comparison of total energy and sojourn time between simulation and analytical models consistently shows excellent agreement across all evaluated cases.

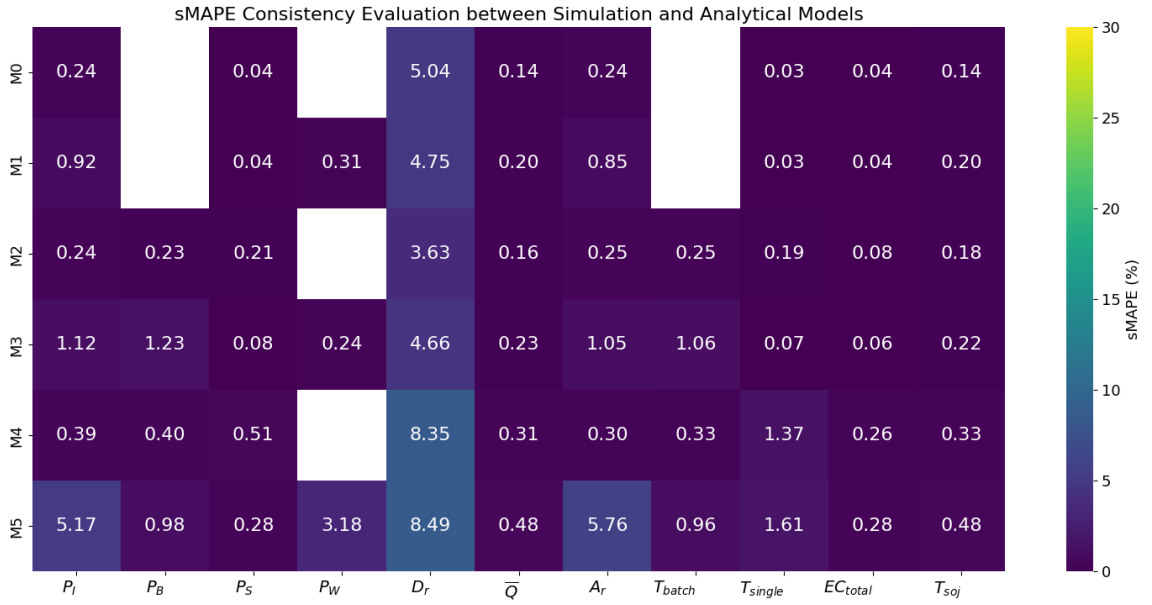

Figure 3.5: Consistency comparison between simulation and analytical models across various performance metrics using sMAPE.

Given the strong consistency observed between the simulation and analytical results we opted to prioritize the analytical approach in the remainder of our experiments. In addition to the accuracy alignment, the analytical approach offers a significantly more time efficient alternative compared to simulation, which often involves repeated executions and longer runtime. Therefore, to optimize both accuracy and computational efficiency, we rely on the CTMC derived metrics in our subsequent performance analysis and model comparisons.

3.4.3. Activation Rate

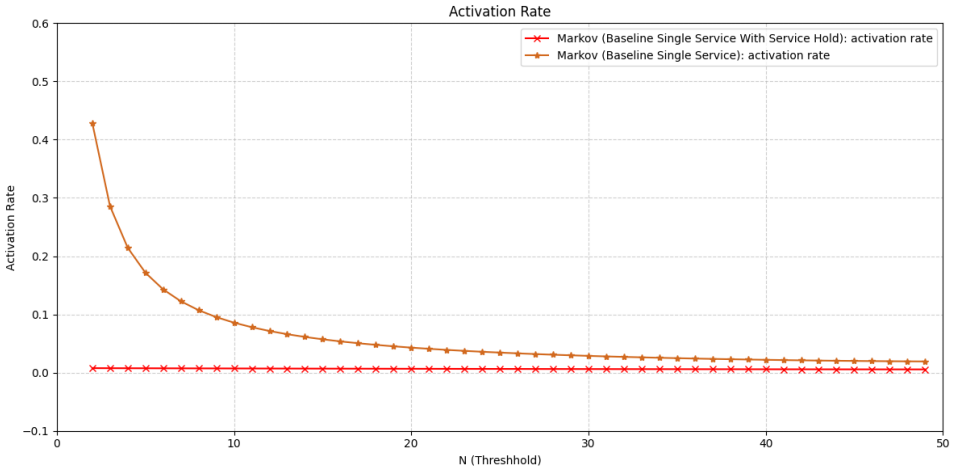

Figure 3.6: Activation rate for baseline single-service models.

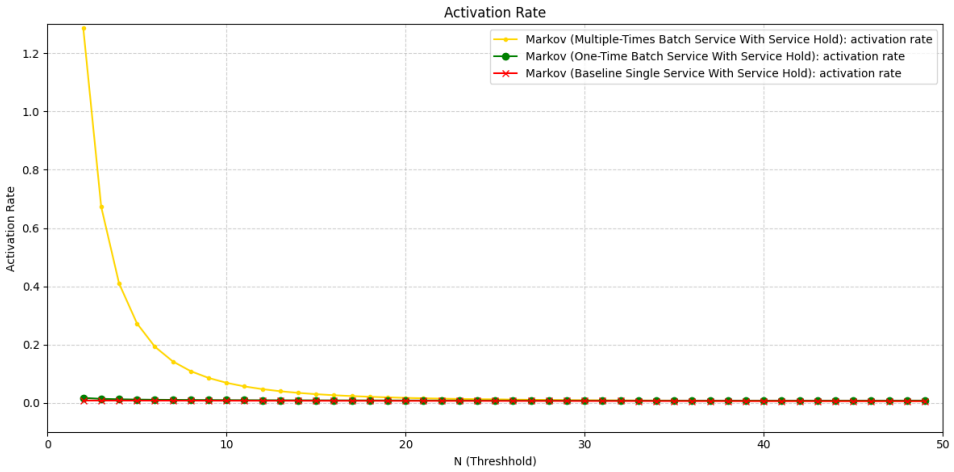

Figure 3.7: Activation rate for models implementing service hold.

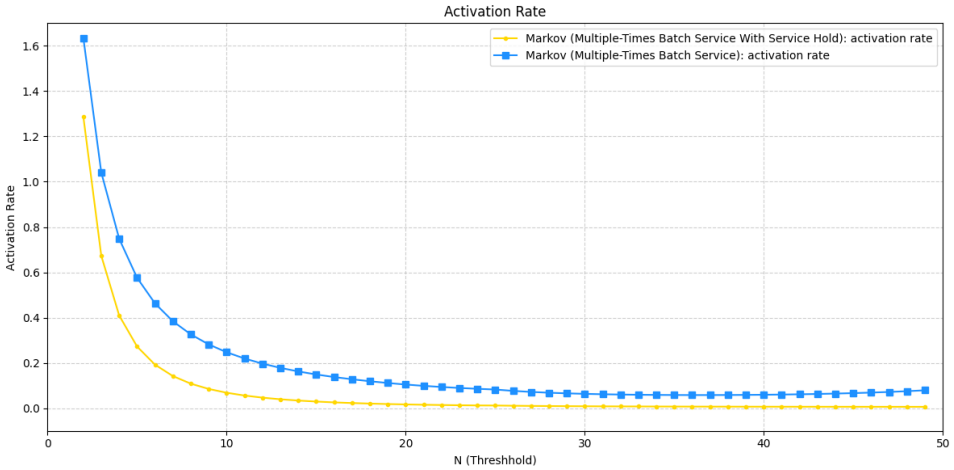

Figure 3.8: Activation rate for multiple times batch service models.

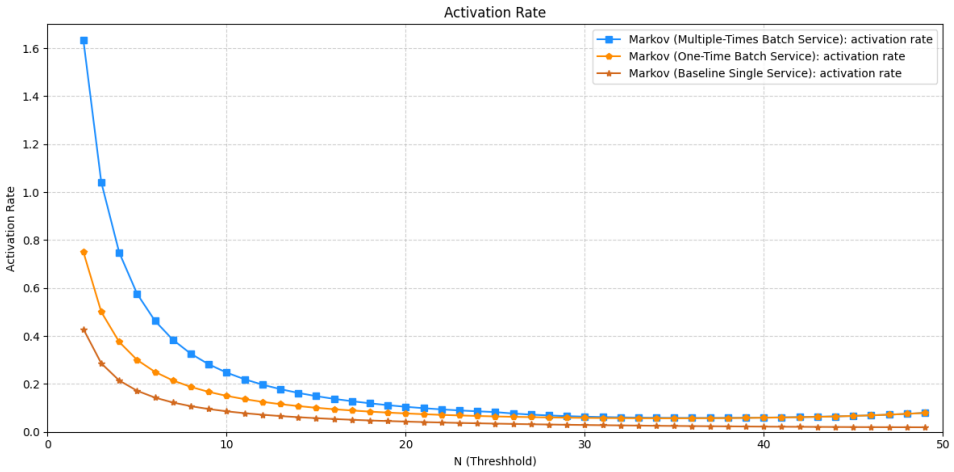

Figure 3.9: Activation rate for models without service hold.

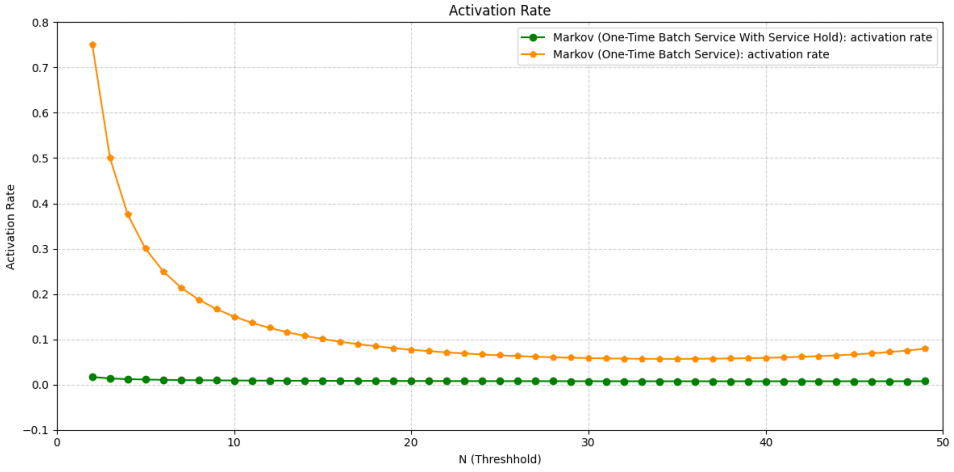

Figure 3.10: Activation rate for one-time batch service models without service hold.

When examining models without service hold mechanisms, a clear inverse relationship emerges between the threshold value N and the activation rate: as N increases, the activation rate decreases. This is observed consistently across the multiple times batch, one-time batch, and baseline single service models. The baseline single service model consistently exhibits the lowest activation rate, followed by the one time batch model, and then the multiple times batch model with the highest rate in the low-threshold range. This order reflects the inherent throughput of each model the higher the throughput, the faster the buffer is emptied, and consequently, the more frequently the node returns to the idle state, increasing activation occurrences. However, as N exceeds approximately half of the buffer capacity, the one-time and multiple times batch models begin to converge in performance, leading to similar activation rates.

When considering the models with service hold, the activation rate drops significantly for all models. Both the baseline single service and one-time batch models with service hold maintain very low activation rates across all threshold values, approaching near-zero levels. While the multiple times batch service with service hold initially shows a higher activation rate compared to its counterparts, it gradually converges to similar values as the threshold increases.

Comparing each model with its respective service hold variant reinforces a consistent trend, service hold mechanisms reduce activation frequency. This is because the wait state delays the transition to the idle state, reducing the number of idle to active cycles. Even for the multiple times batch model, which naturally produces more frequent activations due to its higher throughput, introducing a service hold reduces its activation rate compared to the original version.

In summary, the inclusion of service hold mechanisms offers a substantial reduction in activation rate by limiting unnecessary state transitions. This effect is consistently observed across different queuing policies and becomes particularly beneficial in scenarios where the energy cost associated with node activation is high, thereby significantly influencing the overall energy consumption.

3.4.4. Drop Rate

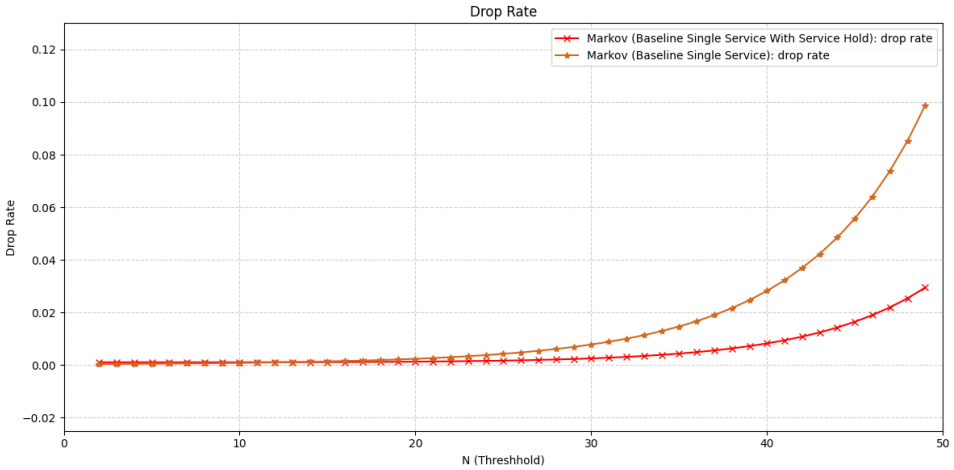

Figure 3.11: Drop rate for baseline single-service models.

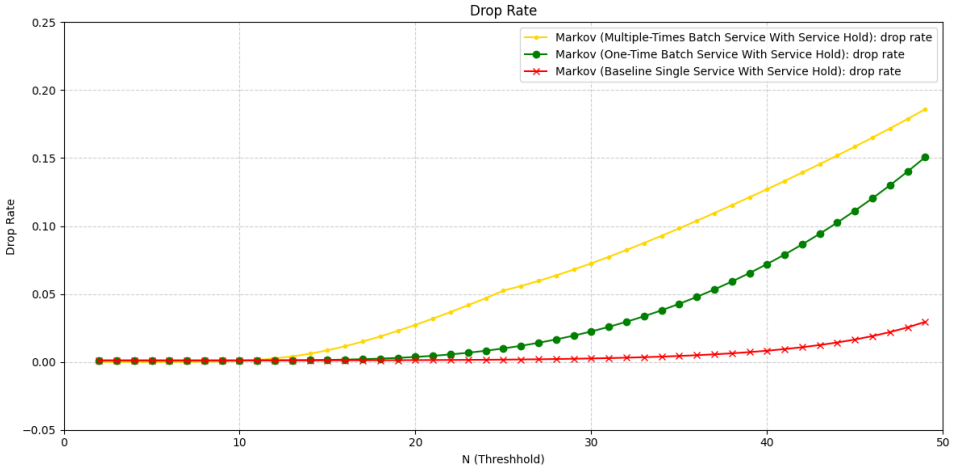

Figure 3.12: Drop rate for models implementing service hold.

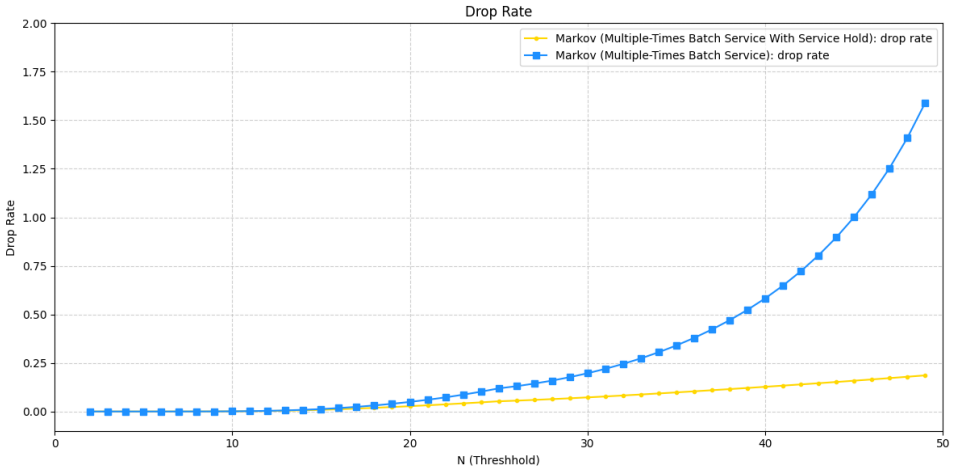

Figure 3.13: Drop rate for the multiple times batch service models.

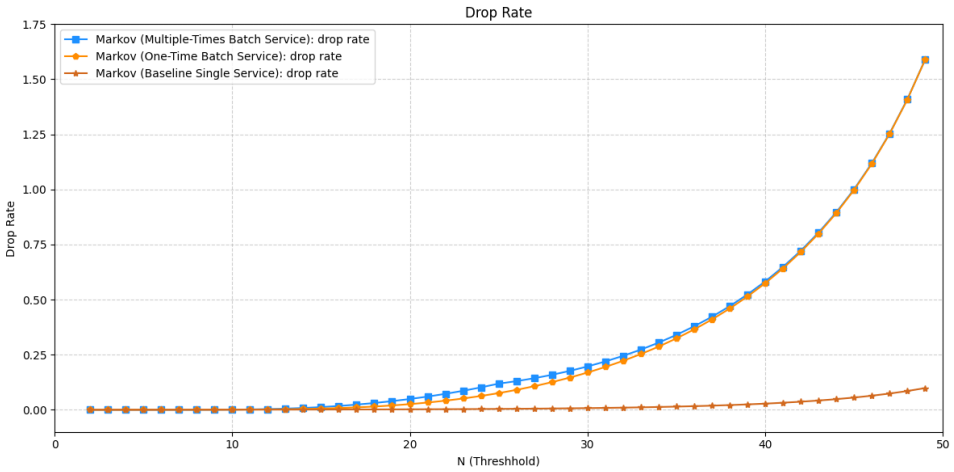

Figure 3.14: Drop rate for models without service hold.

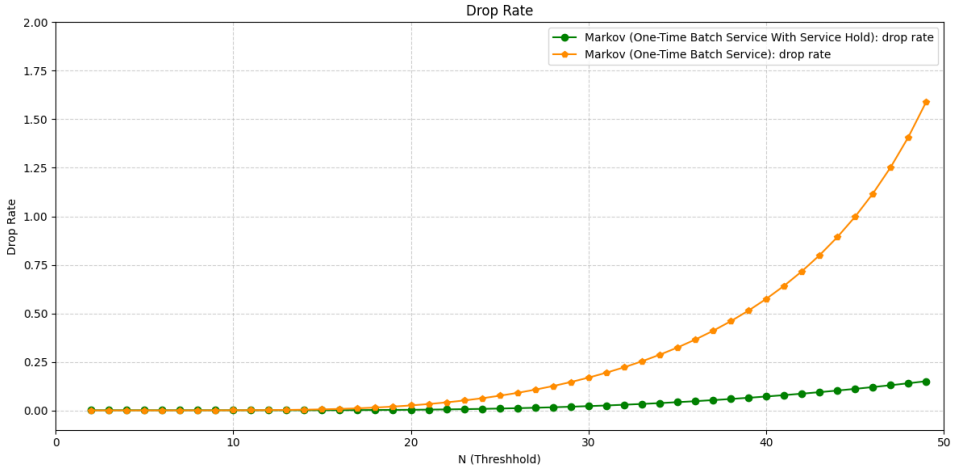

Figure 3.15: Drop rate for the one-time batch service models.

When evaluating models without service hold mechanisms, a distinct pattern emerges in how the drop rate behaves as the threshold value increases. Specifically, models employing batch service exhibit a rising drop rate with increasing threshold N. In contrast, the baseline single service model consistently demonstrates significantly lower drop rates across the threshold range. However, for low threshold values, all three models display similarly low drop rates, suggesting that under such conditions, the buffer rarely reaches its capacity, and packet losses are minimal regardless of the service policy.

This trend persists in the models that incorporate service hold. When comparing the batch-based models with the baseline single-service variant in this category, the one-time and multiple times batch models again present higher drop rates as the threshold increases. Nevertheless, across all three service-hold models, the inclusion of the waiting period generally contributes to a lower drop rate compared to models without service hold. The effect is particularly pronounced in the one-time and multiple times batch models, whereas the difference for the baseline single service model is present but less substantial.

Comparing each model to its corresponding variant with service hold (e.g., the one-time batch with and without service hold) reveals that the benefit of the waiting mechanism in reducing packet loss becomes more apparent as the threshold N increases. For lower values of N, the drop rates between models with and without service hold remain nearly identical. However, as N approaches the buffer capacity K, the service hold mechanism helps alleviate overflow by deferring the transition to the idle state and giving the system more time to process packets. Without service hold, the node quickly enters the idle state after a service phase and waits to accumulate packets up to the threshold before resuming service. When N is large, this delay in processing can lead to rapid buffer overflow, as new arrivals cannot be immediately served. Service hold mitigates this by maintaining the node in a non idle state longer, allowing packets to be processed earlier and reducing the likelihood of drops.

The comparatively higher drop rates observed in batch models, especially as the threshold N approaches the buffer capacity K can be attributed to how packets are handled during batch transmission. In batch service, the node processes N packets as a group, during which no other packets are transmitted. When N is large, this service interval becomes longer, increasing the risk that newly arriving packets will cause the buffer to overflow before the batch transmission ends. In contrast, the baseline single service model continuously serves packets one at a time, enabling it to more quickly clear space in the buffer and accommodate incoming traffic. This ongoing packet handling helps mitigate overflow, particularly under higher arrival rates. Consequently, batch models are more sensitive to buffer saturation as N increases, whereas single service processing maintains steadier control over queue buildup.

3.4.5. Total-Energy

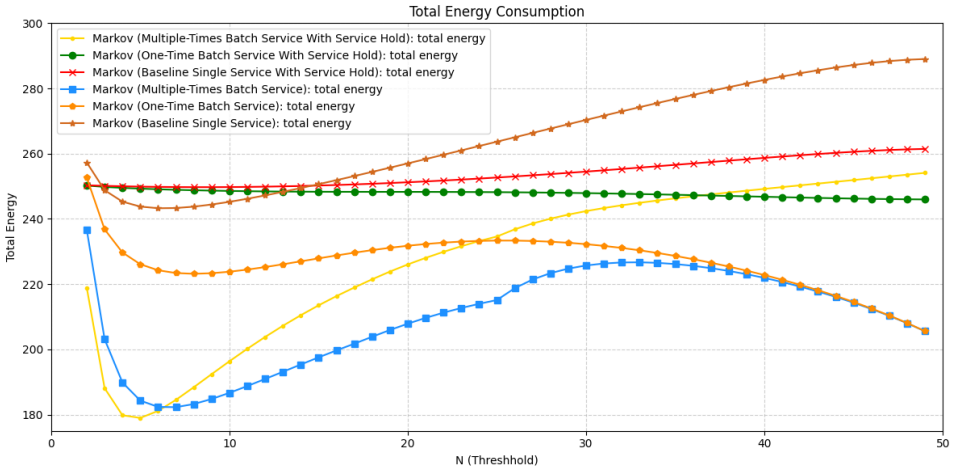

Figure 3.16: Comparison of total energy consumption across all models.

To begin with, the baseline single service model demonstrates better energy efficiency in the initial phase compared to the baseline single service model with service hold and the one-time batch service with service hold. However, as the threshold increases, this advantage diminishes. In particular, when the threshold becomes greater than half the buffer capacity, the total energy consumption of the baseline single service model becomes the highest among all the considered models.

In contrast, both the multiple times batch service with service hold and the multiple times batch service exhibit the lowest energy consumption at the start. The multiple times batch service consistently maintains the lowest energy consumption throughout the entire range of threshold values, making it the most energy-efficient model overall. Nevertheless, the multiple times batch service with service hold begins to consume significantly more energy as the threshold grows, especially beyond the midpoint of the buffer capacity.

The one-time batch service model also shows promising energy-saving performance. Initially, it outperforms the baseline single service model and both baseline single service with service and one time batch with service hold. As the threshold increases, its energy consumption gradually aligns with that of the multiple times batch service model. This convergence suggests that under high threshold configurations, the one-time batch service behaves similarly to the multiple times batch service. Thus, converging to having the second lowest energy consumption.

A crucial insight emerges when examining the impact of service hold. Even though models with service hold tend to reduce the average number of packets in the buffer, thereby reducing holding energy, they introduce a trade-off due to their extended time spent in the active (busy) state. This trade-off becomes critical depending on the platform's hardware. On hardware with a high activation energy cost, models with service hold could offer improved efficiency by reducing the frequency of activations. Conversely, when the activation cost is relatively low, the prolonged busy state can lead to higher overall energy consumption.

Finally, although the drop rate does not significantly affect total energy in our current configuration, it is worth noting that in systems with higher penalties for packet loss, due to retransmission, reliability constraints, or QoS requirements, models with service hold may offer a secondary advantage due to their lower drop rates.

In summary, the multiple times batch service model provides the best energy performance across most configurations. Service-holding strategies show potential under specific hardware constraints, but they introduce a complex trade-off between activation and busy state energy, which must be considered when tailoring the model to a particular application.

3.4.6. Delay

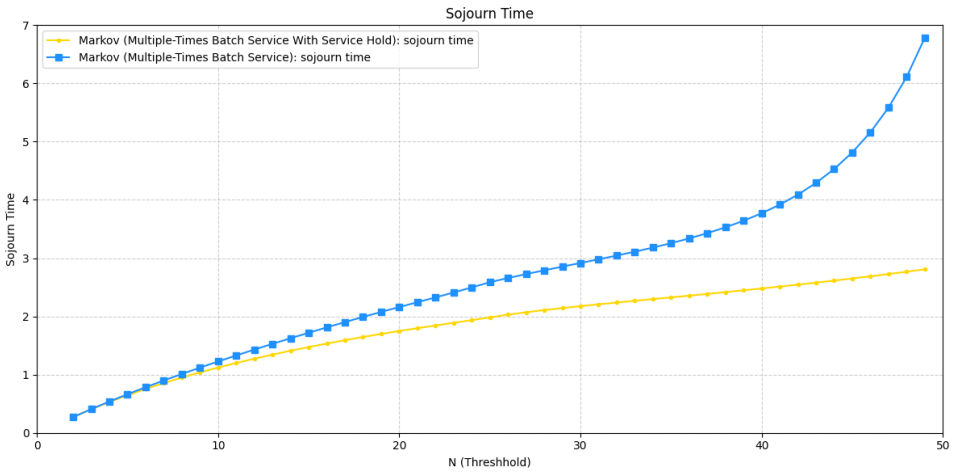

Figure 3.17: Sojourn time for the multiple times batch service models.

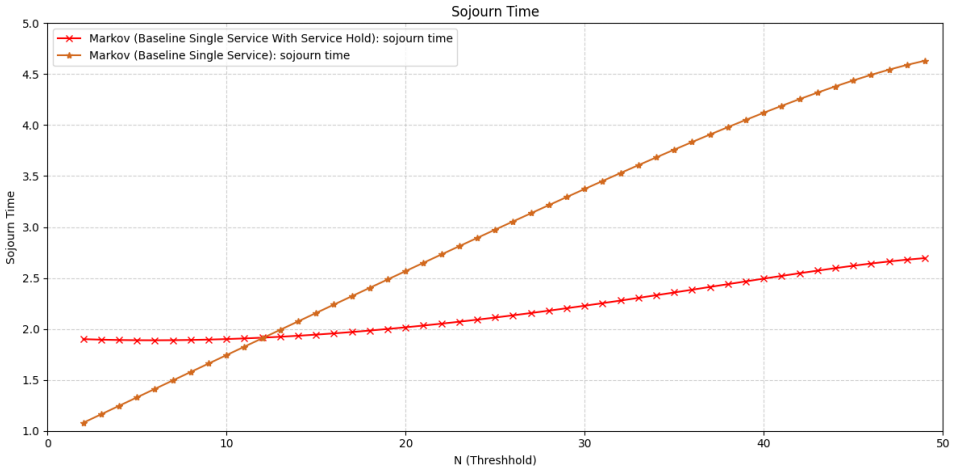

Figure 3.18: Sojourn time for baseline single-service models.

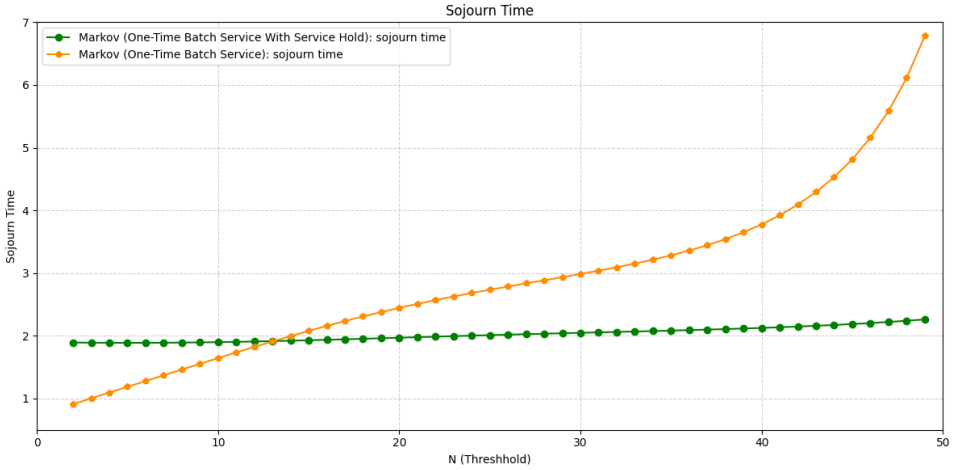

Figure 3.19: Sojourn time for the one-time batch service models.

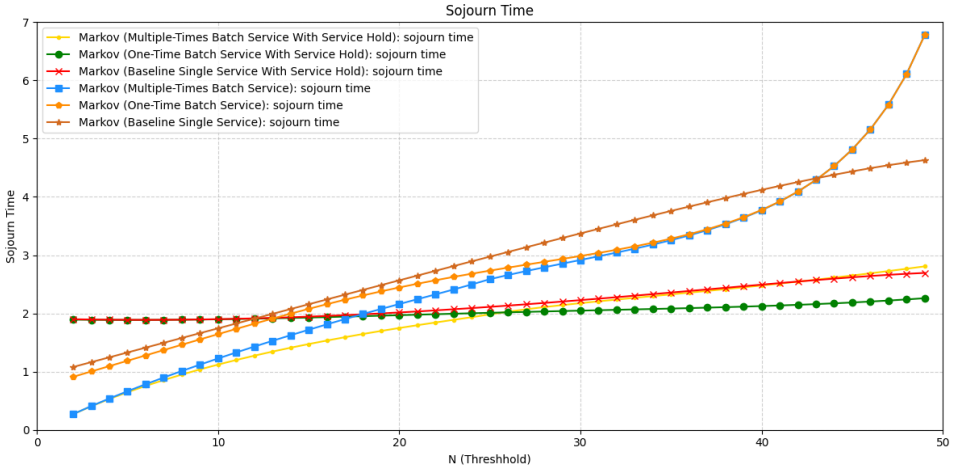

Figure 3.20: Comparison of sojourn times across all service models.

Initially, we observe that the baseline single service model exhibits lower sojourn time compared to the baseline single service with service hold. A similar trend is noted between the one time batch service and the onetime batch service with service hold. In the early stages, models without service hold demonstrate inferior sojourn time, attributable to their immediate service behavior. However, as the threshold increases, the situation reverses: models with service hold maintain a relatively stable sojourn time, while those without it experience a sharp rise. This behavior illustrates the effectiveness of service hold in mitigating excessive queuing delays as the buffer threshold grows.

When evaluating the multiple times batch service versus the multiple times batch service with service hold, the delay behavior differs notably from the previous comparisons. Both models begin with similar sojourn time. Yet as the threshold increases, the delay in the model without service hold increases significantly, while the model with service hold remains more stable and eventually outperforms the other in terms of sojourn time.

Further comparisons among the models with service hold reveal additional insights. At lower threshold values, the multiple times batch service with service hold demonstrates the best delay performance. However, as N increases, its sojourn time also increases, albeit at a manageable rate. Meanwhile, both the baseline single service with service hold and the one-time batch service with service hold start with approximately equal sojourn times. Yet with increasing N, the one time batch service begins to outperform the baseline configuration, maintaining lower delay.

Turning to the models without service hold, the baseline single service consistently exhibits the highest sojourn time across a wide range of threshold values. Only when the threshold approaches the full buffer capacity does its delay become comparable or even slightly better than that of the batch service models. Between the one-time batch service and the multiple times batch service, the latter tends to show slightly better delay performance in the initial phases. However, with increasing threshold, both converge to similar sojourn time values, and ultimately exceed the delay of the baseline single service when nearing buffer saturation.

Overall, a global view of the delay behavior reveals that models incorporating the service hold mechanism generally achieve lower sojourn time as the threshold increases. This outcome stems from the fact that, in models without service hold, packets remain in the buffer longer when the threshold N is large, as the system must wait to accumulate N packets before transitioning to service. Additionally, these models tend to return more frequently to the idle state, further delaying packet service. In contrast, models with the service hold mechanism avoid frequent returns to the idle state and rely on a timeout parameter θ to trigger service.

When N becomes large, the waiting time to reach the threshold in non-service hold models exceeds the timeout duration in service hold models, leading to longer delays. Therefore, models with service hold serve packets more promptly under high-threshold configurations. However, when N is small, the timeout may introduce an unnecessary delay compared to the faster activation based on reaching the threshold, which allows non-service-hold models to outperform in such scenarios.

3.4.7. Energy and Delay Trade-off

To evaluate the performance of different models in terms of both energy efficiency and delay, we introduce a trade-off metric that combines total energy consumption and sojourn time into a single, comparable score. The formula is defined as:

$$\text{Trade\_Off} = \delta \cdot \frac{E}{E_{\text{ref}}} + (1 - \delta) \cdot \frac{W}{W_{\text{ref}}}$$

where E is the total energy consumption and W is the sojourn time. The normalization factors E_{ref} and W_{ref} represent the maximum energy and delay observed across all models under comparison. This normalization ensures that both quantities are dimensionless and scaled equally, enabling fair comparison despite differences in unit or magnitude. The coefficient $\delta \in [0, 1]$ controls the trade-off: values of $\delta < 0.5$ give more weight to delay, $\delta > 0.5$ favor energy minimization, and $\delta = 0.5$ treats both objectives equally. The use of this normalized weighted sum formulation follows a standard scalarization approach in multi-objective optimization, which, as explained by Deb [37], enables the transformation of multiple competing objectives into a single interpretable and tunable performance score when trade-off priorities are known.

The value of δ cannot be fixed universally, as the importance of energy versus delay depends on the application context. For example, in real-time systems like health monitoring, delay is critical and should be prioritized, while in applications such as environmental sensing, energy efficiency may be more important. For this reason, we tested three values: $\delta = 0.5$ (equal importance), $\delta = 0.2$ (delay-prioritized), and $\delta = 0.8$ (energy-prioritized).

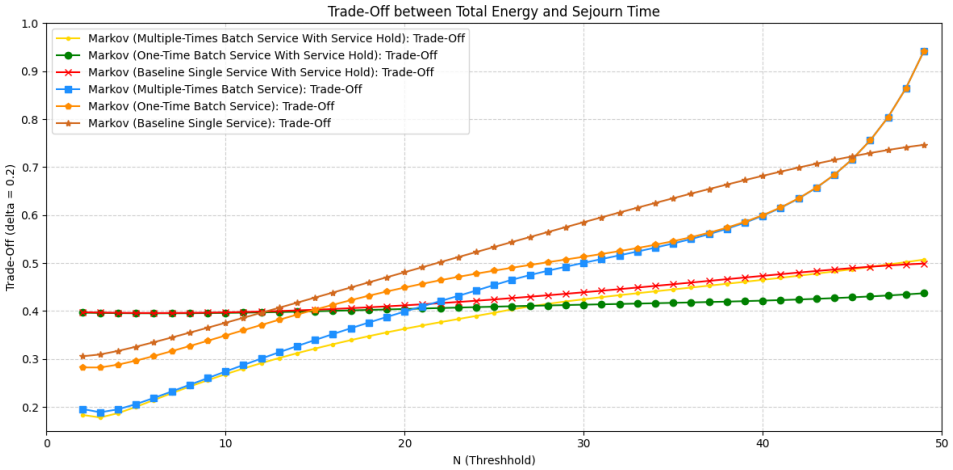

Figure 3.21: Trade-Off between Total Energy and Sojourn Time (δ **= 0.2**)

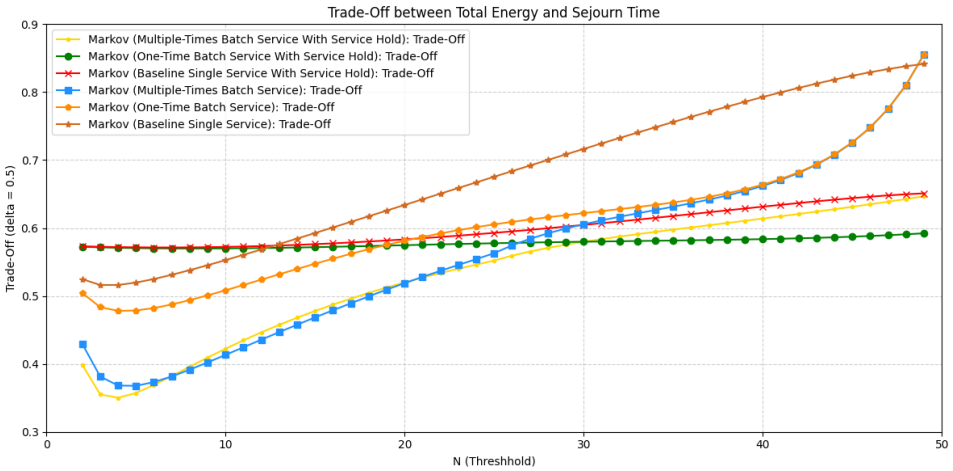

Figure 3.22: Trade-Off between Total Energy and Sojourn Time (δ **= 0.5**)

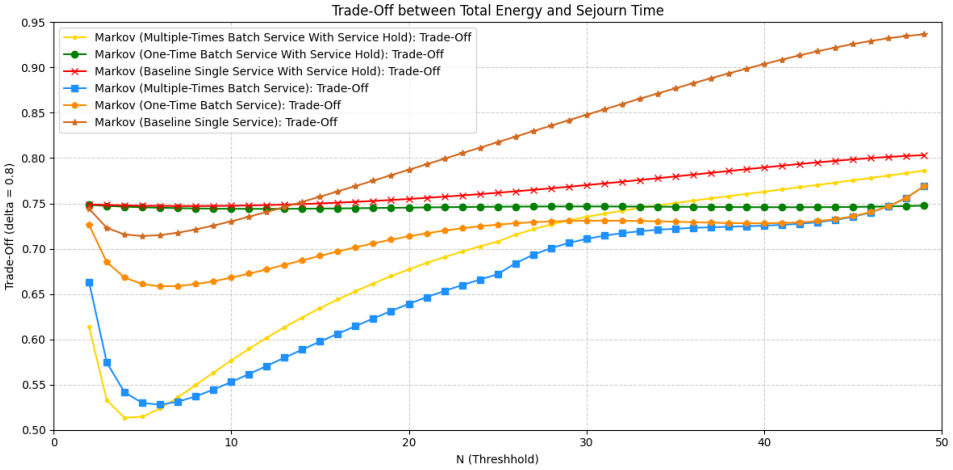

Figure 3.23: Trade-Off between Total Energy and Sojourn Time (δ **= 0.8**)

Before analyzing the results, it is important to note that the trade-off metric is normalized which means all values range from 0 to 1, with lower values indicating better performance. A high trade-off score reflects a poor balance between energy consumption and delay, while a lower score signifies a more efficient model under the given trade-off priorities.

When $\delta = 0.2$, which prioritizes sojourn time over energy consumption, models without service-hold, such as the one-time batch service, multiple-time batch service, and the baseline single service, generally show higher trade-off values. However, this trend is not consistent across all threshold values. For smaller N, particularly before approximately $N = 20$, these models actually perform better than those with service hold. The higher trade-off values for non-service-hold models emerge only as N increases beyond a certain intersection point. The model combining multiple-time batch service with service hold is excluded from this part of the analysis and will be discussed separately.

When $\delta = 0.5$, the trade-off gives equal importance to energy consumption and delay. The overall behavior is similar to the case when $\delta = 0.2$, with models without service hold initially achieving better trade-off values. However, these models tend to reach a lower peak earlier in the threshold range compared to the previous case, indicating slightly better efficiency at small N values. As N increases, the trade-off values of service-hold models improve, and the intersection point, where they begin to outperform non-service-hold models, shifts to higher threshold values. As in earlier cases, the multiple-time batch service with service hold is excluded and will be analyzed separately.

When $\delta = 0.8$, energy consumption becomes the dominant factor in the trade-off. In this configuration, models that implement batch service, whether one-time or multiple-times, with or without service hold, consistently achieve better performance across all threshold values. Their ability to group transmissions into fewer service events reduces overall energy use. In contrast, models with single service consume more energy and thus yield higher trade-off values in this context.

Among all the evaluated models, the multiple-time batch service with service hold consistently demonstrates the most balanced and adaptive behavior. When delay is prioritized, it initially behaves like the non-service-hold models for small N, maintaining low trade-off scores. As N increases, it gradually aligns with the performance of service-hold models, avoiding the performance drop seen in the others. When energy consumption is prioritized, it performs comparably to the other batch-based models, consistently maintaining low trade-off values. In all configurations, it achieves the lowest peak scores across the entire threshold range, making it the most robust and efficient model under varying application priorities.

3.4.8. Wait time-out

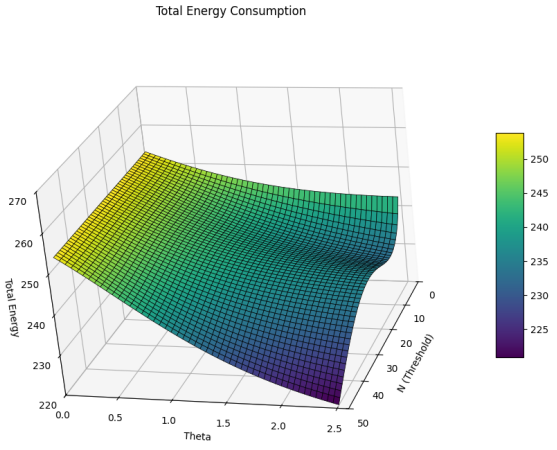

Figure 3.24: 3D comparison of total energy consumption vs. N and θ for one-time batch service with service hold.

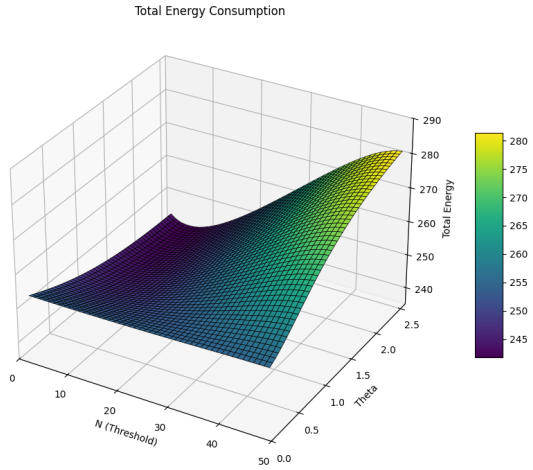

Figure 3.25: 3D comparison of total energy consumption vs. N and θ for Baseline single service with service hold.

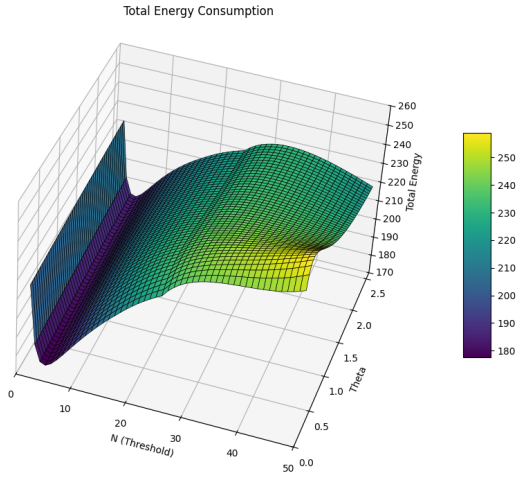

Figure 3.26: 3D comparison of total energy consumption vs. N and θ for multiple-time batch service with service hold.

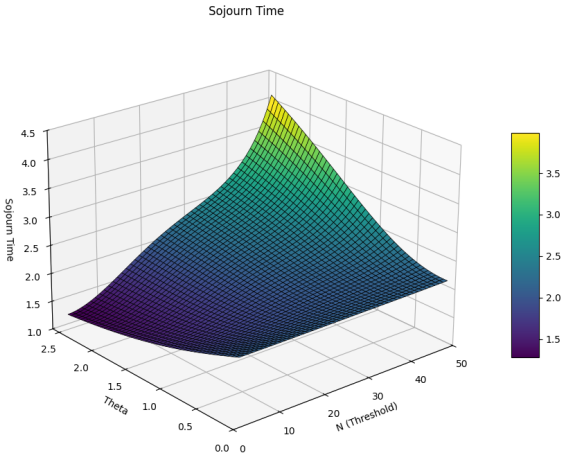

Figure 3.27: 3D comparison of Sojourn time consumption vs. N and θ for one-time batch service with service hold.

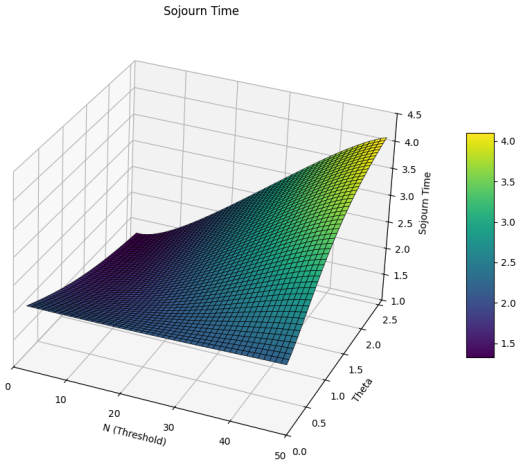

Figure 3.28: 3D comparison of Sojourn time consumption vs. N and θ for Baseline single service with service hold.

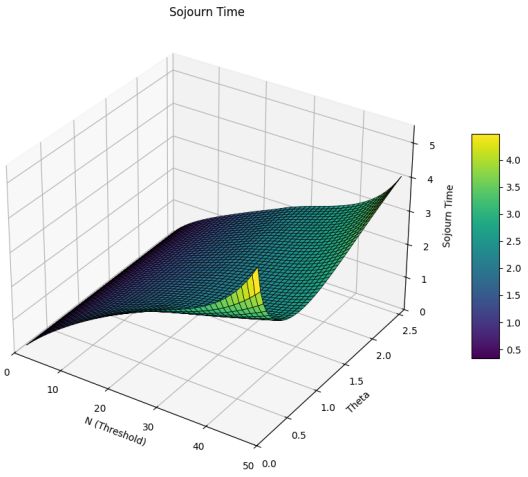

Figure 3.29: 3D comparison of total Sojourn time vs. N and θ for multiple-time batch service with service hold.

In the context of queueing models with service hold, we focus on examining the effect of the parameter θ, where $1/\theta$ denotes the wait timeout. Our primary objective is to analyze how variations in θ influence energy consumption and sojourn time.

In the case of the multiple time batch service with service hold model, variations in θ have little impact on both total energy consumption and sojourn time when the threshold is not approaching the system capacity. However, when the threshold reaches higher values near the system's capacity, a more complex pattern emerges. For total energy, values of θ between 0.1 and approximately 0.25–0.3 lead to a steady increase, reaching a peak, after which further increases in θ result in a gradual decrease in energy consumption. For sojourn time, a similar curved behavior appears, but in the opposite direction: as θ increases, the sojourn time first

decreases and reaches its lowest point around $\theta = 0.5$ to 0.6, and then begins to rise again.

In the case of the one-time batch service with service hold model, there is a clear and consistent inverse relationship between θ and total energy consumption. Across all threshold values, increasing θ results in lower energy usage, and this effect becomes more pronounced as the threshold increases. Regarding sojourn time, for low values of θ, varying the threshold has little impact. However, when θ becomes bigger and the threshold approaches the system capacity, the sojourn time increases significantly, reaching its maximum when both θ and the threshold are at their highest values. As the threshold is reduced under high θ values, the sojourn time decreases accordingly. 00 In the case of the Baseline single service with Service Hold model, both total energy consumption and sojourn time exhibit similar behavior as θ varies. When θ is close to zero, both metrics remain stable across all threshold values. As θ increases, they both decrease when the threshold is low, but increase significantly when the threshold is high.

Conclusion

In this chapter, we validated our simulation models against analytical results using sMAPE across a wide range of performance metrics, including energy consumption, activation rate and drop rate. The results emphasize the importance of selecting queuing policies that align with the specific constraints and objectives of a WSN deployment. Batch service models generally offer better energy efficiency, while incorporating service hold mechanisms can significantly improve delay and reliability. These findings highlight the necessity of flexible model configurations to effectively balance QoS requirements and energy constraints across diverse hardware and application scenarios.

General Conclusion

In this thesis, we addressed the problem of optimizing energy consumption in WSNs while maintaining acceptable levels of communication delay. This challenge stems from the limited energy resources and constrained processing capabilities of sensor nodes, making it crucial to manage node activity intelligently. To tackle this issue, we proposed a modeling and evaluation framework based on queueing theory and formal analysis.

Our contributions include the development of several M/M/1/K queuing models enhanced with N-policy, batch service and service hold . These models were formally represented using CTMCs, from which we derived stationary probabilities to compute key performance metrics. In parallel, we developed a discrete event simulator to implement and validate the models under various configurations and parameter sets.

We first calculated the performance metrics for each model, including drop rate, total energy consumption, sojourn time, and activation rate. We found an excellent match between analytical and simulation results, confirmed quantitatively through the sMAPE metric, which showed strong agreement across all evaluated cases. The experimental results revealed several important insights into the performance of the proposed models. Batch service models consistently demonstrated superior energy efficiency compared to single-packet service approaches, showing significant reductions in energy consumption regardless of the threshold value N. On the other hand, models incorporating a service-hold mechanism exhibited marked improvements in sojourn time and drop rate, particularly under high traffic conditions and larger threshold values. Notably, the model combining multiple-time batch service with service-hold emerged as the most balanced and consistent configuration, offering a favorable energy-delay tradeoff across all tested values of N, without being overly dependent on specific parameter tuning. These findings underscore the complex interdependence between energy consumption, delay, and buffer dynamics, and highlight the importance of adaptive queueing strategies for practical deployment in real-world wireless sensor networks. The 3D analysis further revealed that total energy consumption decreases with lower θ and moderate N, highlighting the sensitivity of batch performance to arrival dynamics. These trends reinforce the

advantage of adaptive batching under varying traffic conditions.

Despite these contributions, the study is not without limitations. The traffic model assumes Poisson arrivals and exponential service times, which may not fully capture the bursty and heterogeneous nature of traffic in real-world WSNs. Moreover, our analysis is constrained to single-node scenarios and does not account for network-wide behaviors such as multi-hop communication, interference, or routing dynamics, which are crucial for large-scale deployments. Additionally, the evaluation is primarily theoretical and simulation-based, without empirical validation in real hardware environments.

For future work, several directions can be explored to enhance and extend our proposed framework. First, researchers may consider optimizing the models that showed promising results particularly the multiple time batch service with service-hold by exploring different parameter configurations and fine-tuning thresholds like N to achieve more favorable tradeoffs under varying network conditions. Second, adapting the model to accommodate more realistic traffic patterns, such as bursty or self-similar arrivals, as well as non-exponential service time distributions, could improve its applicability. Extending the analysis to multi-node systems with cooperative or hierarchical scheduling policies would offer deeper insight into network-wide performance.

In summary, this thesis provides both theoretical and practical foundations for energy-efficient and delay aware design in wireless sensor networks, offering a flexible modeling toolkit that can serve as a basis for future research, enhancement, and real-world deployment.

Appendix A
Evaluating the Practical Feasibility of Batch Service in WSNs

A.1 Definition and Modeling of Batch Service

In the context of this study, batch service refers to a service mechanism where multiple packets are processed together during a single service event. Unlike classical queueing systems that serve packets one by one, a batch service system allows the server to handle a group of B packets in one operation. This is formally represented as a transition in the system's state from n to $n - B$, with $B \geq 1$, where B denotes the batch size (see Figure 0.30).

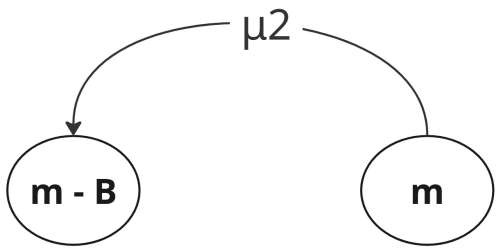

Figure 0.30: Illustration of a Markov chain transition from state m to $m - B$ in batch service

Although this definition is independent of any specific implementation strategy, all the systems or mechanisms we will consider to realize batch service will be modeled using CTMCs. This modeling approach enables a precise description of the system dynamics and facilitates the derivation of key performance measures such as energy consumption and latency.

It is also important to note that the use of batch service in this context is motivated by the potential to reduce operational overhead during packet handling. This may include, for

example, fewer service initiations, reduced channel sensing, or minimized switching between active and idle states. However, at this stage, no specific assumptions are made about which of these factors would be optimized. The goal of the following sections is precisely to examine whether such batch service behavior is technically feasible in wireless sensor networks, and if so, through what kind of mechanisms.

A.2 Investigation of Concurrent Transmission

As part of our effort to identify mechanisms capable of realizing batch service in WSNs, we initially investigated the possibility of concurrent transmission. In this evaluation, we refer to concurrent transmission as the ability of a single sensor node to transmit multiple packets simultaneously, using distinct communication resources such as separate frequency channels, spatial streams, or dedicated radios. This concept is fundamentally different from sequential or time-multiplexed transmission, in that multiple packets are physically transmitted in overlapping time intervals rather than being interleaved or scheduled.

Concurrent transmission was considered as a potential approach because, if feasible, it would naturally enable the behavior we model as batch service (serving multiple packets at once). From a theoretical standpoint, such a mechanism aligns directly with the Markovian modeling of batch service, where the system transitions from state m to state mB in a single step. Hypothetically, this could reduce overall service time per group of packets and might allow more efficient use of the communication channel by aggregating multiple transmissions into one simultaneous event.

Additionally, if implemented effectively, concurrent transmission might improve throughput and possibly reduce latency or energy per packet by shortening the total active transmission duration. These potential advantages made concurrent transmission a relevant and promising direction to explore at the outset of our study.

Having introduced the rationale reasons for investigating concurrent transmission as a potential means to implement batch service, we now turn to a more detailed technical assessment of its feasibility. Specifically, we focus on multi-radio architectures, which represent one of the most direct and theoretically promising ways to enable true physical-layer concurrency within a single sensor node. While the concept of equipping nodes with multiple radios offers the potential for simultaneous transmissions and higher throughput, it also introduces several implementation challenges.

The following analysis, based on our technical review and system-level considerations, highlights the core obstacles encountered in task scheduling, routing, and hardware design. These challenges illustrate why concurrent transmission, despite its conceptual alignment with batch service, is ultimately difficult to realize efficiently in energy-constrained wireless sensor networks.

Task Scheduling

task scheduling in multi-radio Wireless Sensor Networks (MR-WSNs) presents unique challenges due to the interplay between energy efficiency, resource allocation, and dynamic network conditions. Unlike single-radio systems, multiradio nodes must manage concurrent transmissions across multiple transceivers, which introduces complexities in minimizing energy consumption while meeting latency and throughput requirements. The core challenge lies in optimizing task assignments to radios such that idle listening, a significant energy drain, is minimized without overloading any single transceiver. This problem is exacerbated by the heterogeneity of tasks (e.g., transmission bursts, periodic sensing) and the need to balance load dynamically in resource-constrained environments.

A recent study by [38] tackles this by developing an energy consumption model that quantifies both transmission and idle states, then proposes a Particle Swarm Optimization (PSO) algorithm to allocate tasks across radios. While their approach demonstrates improved energy efficiency for moderate workloads, it exposes the fundamental difficulties in multi-radio scheduling, the PSO strategy becomes computationally burdensome beyond 400 tasks, and cannot easily adapt to dynamic network conditions or heterogeneous radio capabilities. These limitations underscore the persistent challenge in multiradio systems: while multiple transceivers theoretically increase throughput, the scheduling overhead and energy trade-offs often negate these benefits in practical deployments.

Routing

Routing in multi-radio WSNs introduces additional complexity compared to traditional single-radio networks, as it requires joint optimization of path selection, radio assignment, and interference management. The core challenge lies in balancing energy efficiency, latency, and throughput while accounting for the dynamic nature of wireless links and the heterogeneity of multi-radio nodes. Unlike single-radio systems, where routing decisions are primarily based on hop count or link quality, multi-radio networks must also consider channel diversity, concurrent transmissions, and the varying power levels of radios. This often leads to conflicting objectives, for example, selecting a high-power radio may reduce hop count but increase interference and energy consumption.

A study by [39] addresses this challenge by formulating routing as an integer linear programming problem that jointly optimizes scheduling, channel assignment, and power control. While their approach demonstrates significant improvements in energy efficiency and end-to-end delay, it reveals critical limitations. The complexity of their solution grows exponentially with network size, making it impractical for large-scale deployments. Additionally, the algorithm assumes static network conditions and requires centralized computation, which is infeasible for dynamic WSNs where topology and link quality fluctuate. These shortcomings highlight the inherent tension in multi-radio routing: while cross-layer optimization can theoretically improve performance, the computational overhead and lack of adaptability often render such solutions impractical for real-world scenarios.

The study also underscores the difficulty of achieving distributed coordination in multi-radio networks. For instance, their proposed random-walk-based distributed algorithm avoids solving complex linear programs but sacrifices optimality, particularly in highly dynamic environments. This trade-off between optimality and scalability remains a key challenge, especially in applications like industrial monitoring or emergency response, where both reliability and adaptability are critical. Future work must explore lightweight, adaptive routing protocols that leverage machine learning or hierarchical optimization to bridge this gap.

Hardware and Cost Constraints

The implementation of multi-radio configurations in modern WSNs continues to face substantial hardware and cost limitations, despite technological advancements. While platforms like the TelosB mote provide a solid foundation for WSN [40] , they are constrained by factors such as power, size, and cost. As noted, the addition of multiple radios typically requires more sophisticated power management and larger PCB designs, which complicates the development and deployment of these systems in compact environments [41]. Furthermore, the design trade-offs between performance, flexibility, power, and size lead to an increase in cost, particularly when considering additional features or sensors in a multi radio node. The cost of sensor nodes must remain moderate to support large-scale deployments, as a significant increase in individual node cost would significantly raise the overall network cost [41].

Complexities in Multi-Radio MAC Design

To understand the limitations of multi radio systems in WSNs, it is essential to first consider the role of *multi-channel MAC protocols*. These protocols were introduced to mitigate interference and improve throughput by distributing communications across different frequency channels. In dense or high traffic WSN deployments, where simultaneous transmissions may

occur across the network, assigning different channels to different nodes or switching dynamically between them helps reduce contention and improve spectral efficiency. Protocols in this space generally fall into three categories: fixed channel assignment, where each node or cluster permanently uses a designated channel; semi dynamic assignment, which allows periodic reallocation of channels; and fully dynamic protocols, which adaptively assign channels in real time based on traffic and interference conditions [42].

Although these techniques have proven effective for inter node communication, they do not provide true concurrent transmission at the node level. That is, while multi-channel MACs allow different nodes to transmit in parallel, they do not enable a single node to transmit multiple packets at the same time. This becomes especially relevant when considering multi radio architectures, where a node is equipped with multiple transceivers. For such a system to function effectively, it must not only have access to multiple channels but also a MAC layer capable of coordinating concurrent activity across those radios.

However, existing multi channel MAC protocols were designed for single-radio environments, and as such, they do not address the intra node coordination, synchronization, and interference management required for multi-radio operation. They cannot dynamically allocate and manage multiple channels across multiple transceivers within a single node while maintaining low overhead and high energy efficiency. This gap in protocol design represents a major obstacle for practical implementation of concurrent transmission in WSNs. Even if the hardware supports multiple radios, the absence of compatible and efficient MAC protocols severely limits the viability of this approach for enabling batch service.

Conclusion of Concurrent Transmission Investigation

Despite its conceptual alignment with batch service, concurrent transmission, specifically through multi-radio systems, presents substantial practical limitations in wireless sensor networks. These include synchronization overhead between transceivers, complex interference management, and high energy demands associated with parallel operation. While theoretically feasible, the coordination and energy costs involved in managing multiple simultaneous transmissions often outweigh the benefits in energy-constrained environments. As a result, concurrent transmission remains an impractical solution for enabling batch service in current WSNs, and further research into efficient protocol design and hardware optimization would be required before it can be seriously considered as a viable alternative.

A.3 Sequential Transmission with TSCH and DSME

Previously, we explored the possibility of implementing batch service in WSNs through concurrent transmission. While the conceptual alignment between concurrent transmission and batch service was clear, practical limitations at the hardware and protocol levels ultimately rendered this approach infeasible. Consequently, we now redirect our investigation toward a more practical direction, the possibility of transmitting multiple packets sequentially within a tightly allocated time window in a single transmission event. This sequential burst, while not simultaneous at the physical layer, still captures the essential characteristic of batch service as defined earlier.

Instead of considering the development of new MAC protocols, we focus on assessing whether the burst behavior required for batch service is already supported by existing standards. Specifically, we examine two operational modes defined in the IEEE 802.15.4 standard — Time Slotted Channel Hopping (TSCH) and Deterministic and Synchronous Multi-channel Extension (DSME). We analyze how these modes enable tightly scheduled transmissions and evaluate their ability to natively support batch service behavior without requiring modifications at the MAC layer.

Time Slotted Channel Hopping

Time Slotted Channel Hopping (TSCH) is a medium access control (MAC) operation mode defined in the IEEE 802.15.4 standard. It provides a deterministic structure to wireless communication by organizing the transmission and reception activities of devices according to a predefined temporal and frequency schedule. In TSCH, each communication opportunity is explicitly assigned in time and frequency, specifying when a device may access the channel, which channel it must use, which peer it communicates with, and the exact timing constraints under which the transmission occurs. This deterministic behavior is enabled through strict synchronization and link scheduling, which collectively transform the wireless medium from a contention-based environment into a coordinated transmission system.

At the core of TSCH is a time-based structure in which time is divided into fixed-length units called timeslots. These timeslots are grouped into slotframes, which define a repeating schedule shared across the network. Each timeslot allows sufficient time for a complete transmission cycle, including a data frame and an optional acknowledgment. The schedule assigns roles to each device for each timeslot, specifying whether a device should transmit, receive, or remain idle. These assignments are encoded in what the standard refers to as links, each associated with specific transmission parameters. This structure is detailed

in [12, Section 6.2.6.1, pp. 69–70]. Through this timeslot allocation mechanism, a node can be scheduled to transmit multiple packets over multiple consecutive timeslots within the same slotframe. This behavior aligns closely with the behavior required by batch service models.

To maintain strict temporal coordination, TSCH employs a global time reference called the Absolute Slot Number (ASN), which increments at each timeslot and is known to all nodes. The ASN not only facilitates synchronization but also plays a key role in channel hopping. In conjunction with the slot specific channel offset, the ASN determines the physical frequency on which each communication occurs. This mechanism spreads transmissions across multiple channels in a pseudo-random pattern, thereby mitigating interference and improving communication reliability. The role of ASN and its associated operations are described in [12, Section 6.2.6.2, p. 69].

Crucially, the reliability of this time-slotted system depends on maintaining synchronization between nodes. TSCH achieves this through a synchronization technique that relies on acknowledgment frames. When a node receives a packet, it can estimate the clock drift relative to the sender and include a timing correction in its acknowledgment. This process ensures that nodes remain aligned within narrow timing tolerances. The precise mechanism for this synchronization, along with the required timing constraints, is defined in [12, Section 6.5.4.3, pp. 95–97].

Another important component of TSCH is its support for multiple link types. Links may be designated as shared, in which multiple nodes compete for transmission rights, or as dedicated, in which a single node is guaranteed exclusive access to the slot. For the purposes of implementing burst transmission, dedicated links are essential. They prevent contention and eliminate the need for backoff or clear channel assessment (CCA), which are otherwise required on shared links. This dedicated mode allows a node to transmit multiple packets in succession, each in its own assigned timeslot, without interruption. The definitions and operational behaviors of these link types are detailed in [12, Section 6.2.5.3, p. 64].

Alltogether, TSCH provides a well-defined framework within the IEEE 802.15.4 standard that supports the implementation of burst transmission. Through its deterministic scheduling, channel diversity, time synchronization, and flexible link allocation, it offers all the necessary mechanisms for orchestrating a series of tightly spaced transmissions by a single node. This structure makes TSCH a suitable and practical candidate for realizing batch service.

Deterministic and Synchronous Multichannel Extension

Deterministic and Synchronous Multichannel Extension (DSME) is a medium access control operation mode defined in the IEEE 802.15.4 standard. It is designed to provide deterministic and collision-free communication by coordinating transmissions over both time and frequency. Unlike purely time-driven mechanisms, DSME is built on top of the beacon-enabled MAC mode, integrating periodic beacons with structured scheduling across multiple channels. Its goal is to support industrial wireless sensor networks and similar applications where bounded latency, energy efficiency, and reliability are essential. DSME achieves this by introducing a multi level time organization and a dynamic slot reservation system that enables exclusive communication opportunities between nodes, without relying on contention-based access.

DSME organizes time using a hierarchical structure composed of superframes and multi superframes. Each superframe begins with a beacon and is divided into two main intervals: the Contention Access Period (CAP) and the Contention-Free Period (CFP). A set of consecutive superframes forms a multi superframe, which repeats over time according to globally defined timing parameters. The duration and repetition of these structures are controlled by MAC-level parameters including macSuperframeOrder (SO), macBeaconOrder (BO), and macMultiSuperframeOrder (MO). These parameters define the lengths of superframes, beacon intervals, and multi-superframe cycles, respectively. Beacons are transmitted periodically at the beginning of each superframe, ensuring network-wide synchronization and enabling devices to align their transmission schedules deterministically. The definitions of these structures and their parameters are detailed in [12, Section 6.11.2.2, pp. 170–173].

Each DSME superframe includes a Contention Free Period (CFP) that contains a configurable number of Guaranteed Time Slots (GTSs). These slots are used for collision-free communication between specific pairs of devices. To acquire one or more GTSs, a device initiates a distributed slot allocation procedure during the preceding Contention Access Period (CAP). This procedure is performed using a three-step handshake composed of the DSME GTS Request, DSME GTS Response, and DSME GTS Notify commands. A device may specify the number of required slots, the preferred superframe index, and the preferred slot ID. Once the allocation is complete, the device is granted exclusive access to the assigned slot(s) during the CFP, free from contention and without relying on CSMA/CA. This behavior enables precise, conflict free scheduling of packet transmissions. The full operation of DSME GTS allocation is specified in [12, Section 6.11.2.4, pp. 174–178].

A key feature of DSME that enables burst transmission is its support for multi-slot GTS allocation. During the GTS request phase, a device may request more than one slot by specifying the number of slots required for its transmission. If granted, these slots may be placed consecutively within the same CFP, or they may be distributed across multiple superframes within a multi superframe cycle. This allocation allows a device to transmit a sequence of packets without interruption, with each packet mapped to a distinct, pre allocated time-frequency slot. As a result, a group of packets can be sent in a tightly controlled burst, consistent with the batch service model described earlier. This form of sequential slot reservation eliminates the need for re-accessing the channel for each packet and guarantees deterministic delivery, provided the GTS allocation succeeds. The capability to perform multi-slot reservation and its associated parameters are detailed in [12, Section 6.11.2.4, pp. 174–178].

DSME enhances communication reliability through its support for channel diversity, which is realized via two operational modes: channel adaptation and channel hopping. In channel adaptation mode, the communication channel for a GTS is selected based on the observed link quality, allowing the system to avoid frequencies experiencing high interference or poor propagation. In channel hopping mode, the frequency used for a GTS varies over time according to a predefined hopping sequence, which is applied consistently across devices. This mechanism distributes transmissions across the spectrum, reducing the likelihood of persistent interference and multipath fading. For burst transmission, both modes improve the robustness of the communication sequence by ensuring that consecutive packets are not all exposed to the same channel conditions. Since each GTS in a multi-slot burst can be mapped to a different frequency, DSME maintains high delivery reliability without compromising determinism. The specifications for channel diversity operation are provided in [12, Section 6.11.2.6, pp. 179–181].

In summary, the DSME operation mode offers sufficient structural and procedural mechanisms to enable burst transmission. The ability to allocate multiple exclusive time-frequency slots, combined with synchronized scheduling and channel management, makes DSME a viable candidate for realizing batch service behaviors in practical WSN deployments.

A.4 Conclusion

This appendix has evaluated the practical feasibility of implementing batch service in wireless sensor networks. While concurrent transmission offers a direct conceptual mapping to batch behavior, it proves impractical due to the complexity and cost of multi-radio architectures. In contrast, the IEEE 802.15.4 standard, through its TSCH and DSME modes, provides native support for tightly scheduled sequential transmissions that effectively approximate batch service. These findings confirm that batch service is not only theoretically but also practically realizable, thereby justifying its use as a foundation for the system model presented in this thesis.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: A survey. *Computer Networks*, 38(4):393–422, 2002.

[2] Sanjeev Ghosh and Srija Unnikrishnan. Reduced power consumption in wireless sensor networks using queue based approach. In *2017 International Conference on Advances in Computing, Communication and Control (ICAC3)*, pages 1–5. IEEE, 2017.

[3] Sanjeev Ghosh and Srija Unnikrishnan. Delay incurred in wireless sensor networks using n-policy m/m/1 queuing model and its mitigation. *International Journal of Electronics, Communications, and Measurement Engineering (IJECME)*, 8(2):40–52, 2019.

[4] VN Jayamani, S Pavai Madheswari, P Suganthi, and SA Josephine. An n-policy m/m/1 queueing model for energy saving mechanism in networks. *Sustainable Computing: Informatics and Systems*, 44:101016, 2024.

[5] Veena Goswami and GB Mund. Prolonging lifetime of wireless sensor networks using modified n policy queueing model. In *Internet of Things and Its Applications: Select Proceedings of ICIA 2020*, pages 335–344. Springer, 2022.

[6] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008.

[7] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, Chichester, UK, 2007.

[8] Dragoljub A Vujic. Wireless sensor networks applications in aircraft structural health monitoring. *Journal of Applied Engineering Science*, 13(2), 2015.

[9] Bingtao Zhang and Lingyan Meng. Energy efficiency analysis of wireless sensor networks in precision agriculture economy. *Scientific Programming*, 2021(1):8346708, 2021.

[10] D. Sharma, Sandeep Verma, and K. Sharma. Network topologies in wireless sensor networks: A review. *International Journal of Electronics and Communication Technology*, 4(3):93–97, 2013.

[11] Bachira Boutoumi and Nawel Gharbi. Two thresholds working vacation policy for improving energy consumption and latency in wsns. In *Queueing Theory and Network Applications: 13th International Conference, QTNA 2018, Tsukuba, Japan, July 25-27, 2018, Proceedings 13*, pages 168–181. Springer, 2018.

[12] IEEE Computer Society. IEEE Standard for Low-Rate Wireless Networks, IEEE Std 802.15.4-2020. IEEE Xplore, 2020. Accessed March 2025.

[13] Kiran Maraiya, Kamal Kant, and Nitin Gupta. Wireless sensor network: a review on data aggregation. *International Journal of Scientific & Engineering Research*, 2(4):1–6, 2011.

[14] Ricardo C Carrano, Diego Passos, Luiz CS Magalhaes, and Celio VN Albuquerque. Survey and taxonomy of duty cycling mechanisms in wireless sensor networks. *IEEE Communications Surveys & Tutorials*, 16(1):181–194, 2013.

[15] Seyed Ali Sharifi. Saving energy and maintaining full coverage in wireless sensor networks with lea3c protocol.

[16] Sultan Mahmood Chowdhury and Ashraf Hossain. Different energy saving schemes in wireless sensor networks: A survey. *Wireless Personal Communications*, 114(3):2043–2062, 2020.

[17] Donald Gross, John F. Shortle, James M. Thompson, and Carl M. Harris. *Fundamentals of Queueing Theory*. Wiley Series in Probability and Statistics. Wiley, Hoboken, NJ, 5th edition, 2018.

[18] Leonard Kleinrock. *Queueing Systems, Volume 1: Theory*. Wiley-Interscience, 1975.

[19] Nai shuo Tian and Zhe George Zhang. Vacation queueing models: Theory and applications. In James H. Dshalalow, editor, *Advances in Queueing: Theory, Methods, and Open Problems*, Operations Research and Management Science, pages 79–104. Springer, 2006.

[20] Daniel P Heyman. The t-policy for the m/g/1 queue. *Management Science*, 23(7):775–778, 1977.

[21] KR Balachandran and Henk Tijms. On the d-policy for the m/g/1 queue. *Management Science*, pages 1073–1076, 1975.

[22] Günter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley-Interscience, Hoboken, NJ, USA, 2nd edition, 2006.

[23] Paul A. Gagniuc. *Markov Chains: From Theory to Implementation and Experimentation*. John Wiley & Sons, 2017.

[24] Jerry Banks, John S Carson II, Barry L Nelson, and David M Nicol. *Discrete-Event System Simulation*. Prentice Hall, 5th edition, 2010.

[25] Vladimir Rykov and Dmitry Kozyrev. On sensitivity of steady-state probabilities of a cold redundant system to the shapes of life and repair time distributions of its elements. In *International Workshop on Simulation*, pages 391–402. Springer, 2015.

[26] Jie Chen, Biplab Sikdar, and Mounir Hamdi. An adaptive n-policy queueing system design for energy efficient and delay sensitive sensor networks. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[27] Changzhen Zhang, Jun Yang, and Ning Wang. Timely reliability modeling and evaluation of wireless sensor networks with adaptive n-policy sleep scheduling. *Reliability Engineering & System Safety*, 235:109270, 2023.

[28] Fuu-Cheng Jiang, Der-Chen Huang, Chao-Tung Yang, and Fang-Yi Leu. Lifetime elongation for wireless sensor network using queue-based approaches. *The Journal of Supercomputing*, 59:1312–1335, 2012.

[29] Der-Chen Huang, Ying-Yi Chu, and Wei-Ming Chen. A life extend approach based on multiple priority queue n strategy for wireless sensor network. *Journal of Internet Technology*, 20(2):457–470, 2019.

[30] Bachira Boutoumi and Nawel Gharbi. An energy saving and latency delay efficiency scheme for wireless sensor networks based on gspns. In *2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 0645–0650. IEEE, 2017.

[31] Crossbow Technology, Inc. TelosB Mote Platform Datasheet. `https://web.archive.org/web/20080510034651/http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/TelosB_Datasheet.pdf`, 2007. Accessed via Internet Archive.

[32] Crossbow Technology, Inc. MICAz Mote Platform Datasheet. `https://web.archive.org/web/20080509122535/http://www.xbow.com/`

Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf, 2006. Accessed via Internet Archive.

[33] Quentin Bramas, Wilfried Dron, Mariem Ben Fadhl, Khalil Hachicha, Patrick Garda, and Sébastien Tixeuil. Wisebat: accurate energy benchmarking of wireless sensor networks. In *2015 Forum on Specification and Design Languages (FDL)*, pages 1–8. IEEE, 2015.

[34] Christos Antonopoulos, Fotis Kerasiotis, Christos Koulamas, George Papadopoulos, and Stavros Koubias. Experimental evaluation of the waspmote platform power consumption targeting ambient sensing. In *2015 4th Mediterranean Conference on Embedded Computing (MECO)*, pages 124–128. IEEE, 2015.

[35] J. Scott Armstrong. *Long-Range Forecasting: From Crystal Ball to Computer*. John Wiley & Sons, 2nd edition, 1985.

[36] Rob J. Hyndman and Anne B. Koehler. Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679–688, 2006.

[37] Kalyanmoy Deb. Multi-objective optimization using evolutionary algorithms. john wiley & sons ltd. *Baffins Lane, Chichester, West Sussex, PO19 1UD*, 2001.

[38] Qiao Yan, Wei Peng, and Guiqing Zhang. Optimal energy consumption tasks scheduling strategy for multi-radio wsns. *Sensors*, 20(3):881, 2020.

[39] Jinbao Li, Xiaohang Guo, Longjiang Guo, Shouling Ji, Meng Han, and Zhipeng Cai. Optimal routing with scheduling and channel assignment in multi-power multi-radio wireless sensor networks. *Ad Hoc Networks*, 31:45–62, 2015.

[40] Michael Johnson, Michael Healy, Pepijn Van de Ven, Martin J Hayes, John Nelson, Thomas Newe, and Elfed Lewis. A comparative review of wireless sensor network mote technologies. *SENSORS, 2009 IEEE*, pages 1439–1442, 2009.

[41] Fatma Karray, Mohamed W Jmal, Alberto Garcia-Ortiz, Mohamed Abid, and Abdulfattah M Obeid. A comprehensive survey on wireless sensor node hardware platforms. *Computer Networks*, 144:89–110, 2018.

[42] Rana Diab, Gérard Chalhoub, and Michel Misson. Overview on multi-channel communications in wireless sensor networks. *Netw. Protoc. Algorithms*, 5(3):112–135, 2013.