UNIVERSITY OF SALENTO

Detailed Analysis of Renewable Energy-Based Desalination System for Maritime Vessels

PROFESSORS:
Gianfranco PARLANGELI

SUPERVISOR:
Carlo
DE GIORGI

STUDENT:
Sara FARRIS
Danilo GIOVANNICO

Academic year 2024

Sommario

1. Abstract

This report presents a detailed analysis of a network of smart buoys designed to produce electricity from renewable sources and desalinate water. Serving as replenishment points for maritime vessels, these buoys provide both desalinated water and energy. Given the significant shortage of infrastructure offering these services to small vessels, the mobile and environmentally friendly buoys offer a flexible solution, suitable for installation even in protected natural parks. With a growing demand for maritime services, the study aims to optimize energy production and water desalination management for efficient resource use. Various strategies for optimization and storage are explored to develop a resilient fleet of buoys capable of robustly supporting all vessel needs, reducing the reliance on permanent infrastructure.

2. Introduction

The increasing demand for sustainable and autonomous systems in marine environments has driven the development of buoys equipped with renewable energy sources and desalination capabilities. These buoys provide small vessels with essential services typically available at dockside facilities, but with greater flexibility and less environmental impact. Addressing the shortage of infrastructure for water and electricity supply, these mobile systems can be deployed even in protected natural parks.

A significant advantage of this approach is the use of a multi-agent system, where buoys can specialize in either water production or energy storage. This specialization, combined with optimization algorithms, ensures optimal resource management and storage. The system can dynamically allocate vessels to the appropriate buoys, maintaining a consistent and robust service. By analyzing strategies for optimization and storage, the study aims to create an efficient and reliable network of buoys, capable of meeting the demands of maritime vessels and promoting sustainable operations in marine environments.

3. System Description

Each buoy is equipped with the following capabilities:

- **Energy Production System**: Converts renewable energy sources into electricity, denoted as Cr, with a maximum generation capacity of 4.6 kWh (10 hours solar and 4 hours wind).
- **Battery**: Stores the generated electricity with a maximum capacity of 5 kWh, strongly limited by the buoy's maximum weight. The target battery level is set at 80% to maximize battery lifespan.
- **Desalination System**: Uses stored energy to desalinate water with an efficiency rd of 0.005 L/kWh, converting energy into water up to a maximum storage capacity of 10 liters, also constrained by the buoy's weight limit.

The primary components of each buoy are summarized in the table below:

Description	Symbol	Quantity	Unit of Measure
Maximum buoy battery capacity	-	5	kWh
Maximum buoy water tank capacity	-	10	liters
Maximum solar panel power	-	300	W

Description	Symbol	Quantity	Unit of Measure
Maximum wind generator power	-	400	W
Maximum recharge from renewable source	Cr	4.6	kWh
Water level	x_2^{target}	100	%
Battery level	x_1^{target}	80	%
Maximum energy request per vessel u1	-	3	kWh
Maximum water request per vessel u1	-	10	liters
Energy/water conversion efficiency	1/kc	0.005	L/kWh

Each buoy has an external fiberglass structure with a diameter of half a meter and a depth of about 5 meters, forming a cylindrical shape. The system uses hardware already established in the maritime field and, in this implementation, employs a reverse osmosis desalination system due to its high productivity. Other desalination systems can be implemented to avoid the production of wastewater.

Groups of buoys will also be equipped with appropriate communication systems to ensure connectivity among buoys within the same buoy field (typically Bluetooth or other short-range communication systems) and long-range communication systems to enable information exchange between multiple buoy fields.

The requests from the vessels at each instant k occur randomly in terms of both the number of requests and the maximum resources to be requested. A vessel cannot request a number of resources that falls below the minimum threshold that each buoy can offer. Specifically, a vessel can request up to a maximum of 3 kWh of energy to recharge its onboard batteries and up to 10 liters of desalinated water, often taking whatever is available.

4 State of the Buoys and Hypotheses

4.1 State of Buoy

4.1.1 Components and Functionality

Each buoy in the network is designed to perform the following functions:

1. **Energy Production:**
   - The buoy is equipped with a system that produces electricity from renewable sources, such as solar or wind energy.
   - This production is represented by the constant c_r, which denotes the amount of renewable energy generated per time step.
2. **Energy Storage:**
   - The generated energy is stored in a battery.
   - The battery's discharge efficiency is denoted by r_s. This efficiency impacts how much stored energy is effectively available for use.
3. **Water Desalination:**
   - The buoy uses stored energy to desalinate water.
   - The desalination process efficiency is represented by r_d, indicating the conversion rate of energy to desalinated water.

4.1.2 State Variables

The state of each buoy is described by two primary state variables:

- x_1: Represents the accumulated energy in the buoy's battery, measured in kilowatt-hours (kWh).
- x_2: Represents the accumulated desalinated water, measured in liters (L).

4.2 Hypotheses

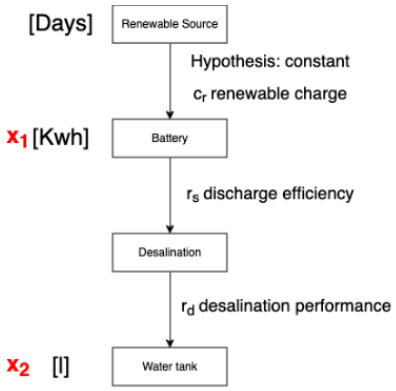
The hypotheses set the framework and priorities for the system's operation:

1. **Prioritization of Water Production:**
   - The system gives priority to the production of desalinated water. This means that energy management and allocation will be primarily directed towards maximizing water output.
2. **Target Values:**
   - The goal is to reach specific target values for the state variables:
     - x_1^{target}: The desired amount of accumulated energy.
     - x_2^{target}: The desired amount of accumulated desalinated water.

4.3 Summary

The state of the buoy, represented by the accumulated energy (x_1) and accumulated desalinated water (x_2), evolves over time according to the dynamic equations. The hypotheses prioritize water production and aim to reach specific target values for energy and water. When vessels interact with the buoys by withdrawing resources, the dynamic equations are adjusted to reflect these interactions. This detailed understanding helps in effectively managing and optimizing the production and utilization of resources in the buoy network.

5 Energy Flow Analysis



State	x_1 x_2	Accumulated energy [kwh] Accumulated water [l]
Input	u_1 u_2	Energy required Water required

$x_1^{percentuale} = (x_1 / \text{Maximum charge}) \cdot 100$

$x_2^{percentuale} = (x_2 / \text{Maximum charge}) \cdot 100$

$$x_1 \cdot r_s \cdot r_d = x_2$$

$$1/k_c$$

$$[Kwh] \longrightarrow [l]$$

The energy flow analysis describes the transformation and flow of energy through the system, starting from renewable sources and ending in the accumulation of desalinated water in the tank.

5.1 Components and Process

1. Renewable Source ([Days] to [kWh]):
    a. The process begins with a renewable energy source, such as solar panels or wind turbines.
    b. Hypothesis: The renewable energy charge (c_r) is constant, indicating a steady input of renewable energy over time.
2. Energy Storage (Battery [kWh]):
    a. The generated renewable energy (x_1 measured in kilowatt-hours [kWh]) is stored in a battery.
    b. Efficiency Factor: The battery has a discharge efficiency represented by r_s. This factor determines the effective energy available for use after accounting for storage losses.
3. Desalination Process ([kWh] to [L]):
    a. The stored energy is used to power the desalination system, which converts seawater into desalinated water.
    b. Efficiency Factor: The desalination process has a performance efficiency denoted by r_d. This factor indicates the effectiveness of energy conversion into desalinated water.
4. Water Accumulation ([L]):
    a. The output of the desalination process is accumulated in a water tank.
    b. The accumulated water is represented by x_2, measured in liters [L].

5.2 Energy Conversion Formula

The energy flow diagram encapsulates the conversion of energy from one form to another through the system:

$$x_1 \cdot r_s \cdot r_d = x_2$$

Where:

- x_1: Accumulated energy in kWh.
- r_s: Battery discharge efficiency.
- r_d: Desalination performance efficiency.
- x_2: Accumulated desalinated water in liters.

This can be rewritten as:

$$x_2 = \frac{x_1}{k_c}$$

Where k_c is the combined conversion factor from energy to water, encompassing both r_s and r_d.

5.3 Considerations for Efficiency

The overall efficiency of converting renewable energy into desalinated water depends on the combined efficiencies of the battery (r_s) and the desalination process (r_d). The state variables (x_1 and x_2) represent the effective energy and water quantities after accounting for these efficiencies.

5.4 Summary

The energy flow analysis provides a comprehensive overview of how renewable energy is harnessed, stored, and converted into desalinated water. By understanding the efficiencies at each stage, one can optimize the system to ensure maximum water production with the available energy resources. This analysis is crucial for effective management and utilization of renewable energy in the desalination process, ensuring that the buoys can provide a reliable supply of both energy and water to vessels in need.

6 Dynamic Equations

Each buoy is characterized by the state variables x_1 (level of charge) and x_2 (level of liquid). These states are increased thanks to renewable energy production and consumed by the demands of the vessels.

- **State Variables:**
  - x_1: Accumulated energy in kWh
  - x_2: Accumulated water in liters
- **Input Variables:**
  - u_1: Energy required
  - u_2: Water required

The system prioritizes water production aiming to reach target values x_1^{target} and x_2^{target}.

Hypothesize

- We hypothesize using a ΔT equal to 1 day to consider all charging and discharging phenomena as punctual and instantaneous.
- Identical Buoys: All buoys are structurally identical.
- Constant Renewable Production: Renewable energy production is constant at each time step.
- Simulation Step k: Initial simulations use a time step k to consider charging and discharging phenomena as instantaneous.
- Hourly Simulation Step: Subsequently, simulations use a time step k=1 hour to account for variable vessel traffic throughout the day.
- Resource Prioritization: The system prioritizes the scarcer resource, water production, ensuring reintegration if sufficient charge is present.

Given the following variables:

- x_1 represents the accumulated energy.
- x_2 represents the accumulated water.
- c_r is the constant amount of renewable energy available.
- k_c is an energy-to-water conversion parameter.

The system prioritizes water production, aiming to reach target values $\mathbf{x_1^{target}}$ and $\mathbf{x_2^{target}}$. Two cases are considered, we can therefore analyze the dynamic equations:

6.1 Case 1 - Insufficient Battery Capacity

If the battery cannot recharge the entire tank, represented by the inequality:

$$\frac{x_1}{k_c} < (x_2^{\text{target}} - x_2)$$

The dynamic equations are:

$$\begin{cases} x_1(k+1) = x_1(k) + c_r - \frac{k_c x_1(k)}{k_c} \\ x_2(k+1) = x_2(k) + \frac{x_1(k)}{k_c} \end{cases}$$

In matrix form:

$$x(k+1) = A_{sc}x(k) + C_{sc}$$

$$x(k+1) = \begin{bmatrix} 0 & 0 \\ \frac{1}{k_c} & 1 \end{bmatrix} x(k) + \begin{bmatrix} c_r \\ 0 \end{bmatrix}$$

Resulting in:

$$\begin{cases} x_1 = c_r \\ x_2 < x_2^{\text{target}} \end{cases}$$

6.2 Case 2 - Sufficient Battery Capacity

If the battery can recharge the entire tank:

$$\frac{x_1}{k_c} > (x_2^{\text{target}} - x_2)$$

The dynamic equations are:

$$\begin{cases} x_1(k+1) = x_1(k) + c_r - k_c(x_2^{\text{target}} - x_2(k)) \\ x_2(k+1) = x_2^{\text{target}} \end{cases}$$

In matrix form:

$$x(k+1) = A_{si}x(k) + C_{si}$$

$$x(k+1) = \begin{bmatrix} 1 & +k_c \\ 0 & 0 \end{bmatrix} x(k) + \begin{bmatrix} c_r - k_c x_2^{\text{target}} \\ x_2^{\text{target}} \end{bmatrix}$$

Resulting in:

$$\begin{cases} x_1 \neq 0 \\ x_2 = x_2^{\text{target}} \end{cases}$$

6.3 Complete Case

Combining both scenarios, the complete model accounts for the state of multiple buoys:

$$\begin{bmatrix} \begin{bmatrix} x_1^1 \\ x_2^1 \end{bmatrix} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \begin{bmatrix} x_1^n \\ x_2^n \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} [A_{sc}^1 \text{ or } A_{si}^1] & \cdot & \cdot \\ \cdot & [A_{sc}^2 \text{ or } A_{si}^2] & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & [A_{sc}^n \text{ or } A_{si}^n] \end{bmatrix} \begin{bmatrix} \begin{bmatrix} x_1^1(k) \\ x_2^1(k) \end{bmatrix} \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \begin{bmatrix} x_1^n(k) \\ x_2^n(k) \end{bmatrix} \end{bmatrix} + \begin{bmatrix} [C_{sc}^1 \text{ or } C_{si}^1] \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ [C_{sc}^n \text{ or } C_{si}^n] \end{bmatrix}$$

- $\mathbf{A_{sc}}$: Complete discharge scenario where all available energy is used.
- $\mathbf{A_{si}}$: Incomplete discharge scenario where only part of the available energy is used.
- $\mathbf{C_{sc}}$, $\mathbf{C_{si}}$: Vector of constant terms represents the fixed contributions of energy and water.

6.4 Considerations for Vessels

When integrating the interaction of vessels (such as boats) that withdraw resources from the buoys, additional variables and adjustments are introduced:

- **Energy Withdrawal (u_1):** The amount of energy taken by the vessel.
- **Water Withdrawal (u_2):** The amount of water taken by the vessel.

The dynamic equations are modified to account for these withdrawals:

6.4.1 Case 1 - Insufficient Battery Capacity

1. **Energy Evolution with Withdrawal:**

$$x_1(k+1) = c_r - u_1$$

2. **Water Evolution with Withdrawal:**

$$x_2(k+1) = x_2(k) + \frac{x_1(k)}{k_c} - u_2$$

In matrix form:

$$x(k+1) = \begin{bmatrix} 0 & 0 \\ \frac{1}{k_c} & 1 \end{bmatrix} x(k) + \begin{bmatrix} c_r - u_1 \\ -u_2 \end{bmatrix}$$

6.4.2 Case 1 - Sufficient Battery Capacity

1. **Energy Evolution with Withdrawal:**

$$x_1(k+1) = x_1(k) + c_r - k_c(x_2^{\text{target}} - x_2(k)) - u_1$$

2. **Water Evolution with Withdrawal:**

$$x_2(k+1) = x_2^{\text{target}} - u_2$$

In matrix form:

$$x(k+1) = \begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 1 & +k_c \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} c_r - k_c x_2^{\text{target}} - u_1 \\ x_2^{\text{target}} - u_2 \end{bmatrix}$$

The full model could then include these withdrawals in the dynamic equations:

$$\begin{bmatrix} \begin{bmatrix} x_1^1 \\ x_2^1 \end{bmatrix} \\ . \\ . \\ . \\ . \\ \begin{bmatrix} x_1^n \\ x_2^n \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} [A_{sc}^1 \text{ or } A_{si}^1] & . & . \\ . & [A_{sc}^2 \text{ or } A_{si}^2] & . \\ . & . & . \\ . & . & . \\ . & . & . \\ . & . & . \\ . & . & [A_{sc}^n \text{ or } A_{si}^n] \end{bmatrix} \begin{bmatrix} \begin{bmatrix} x_1^1(k) \\ x_2^1(k) \end{bmatrix} \\ . \\ . \\ . \\ . \\ \begin{bmatrix} x_1^n(k) \\ x_2^n(k) \end{bmatrix} \end{bmatrix} + \begin{bmatrix} [C_{sc}^1 \text{ or } C_{si}^1] \\ . \\ . \\ . \\ . \\ [C_{sc}^n \text{ or } C_{si}^n] \end{bmatrix} + \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ . \\ . \\ . \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

summarizing in detail:

State of the Buoys

Each buoy has two state variables:

- x_1^i: Energy accumulated in buoy i at time k.
- x_2^i: Water accumulated in buoy i at time k.

These variables are collected in a column vector for each buoy:

$$\begin{bmatrix} x_1^i \\ x_2^i \end{bmatrix}$$

Complete System of Buoys

The complete system representation with n buoys is:

$$\begin{bmatrix} \begin{bmatrix} x_1^1 \\ x_2^1 \end{bmatrix} \\ . \\ . \\ . \\ . \\ \begin{bmatrix} x_1^n \\ x_2^n \end{bmatrix} \end{bmatrix}$$

Transition Matrix

The system dynamics are described by a transition matrix that can vary depending on whether there is a complete or incomplete discharge of the accumulated energy. This matrix is composed of sub-matrices $\mathbf{A_{sc}^i}$ or $\mathbf{A_{si}^i}$, where:

- $\mathbf{A_{sc}}$: Represents the case of complete discharge.
- $\mathbf{A_{si}}$: Represents the case of incomplete discharge.

The complete transition matrix is:

$$\begin{bmatrix} \begin{bmatrix} [A_{sc}^1 \text{ or } A_{si}^1] & . & . \\ . & [A_{sc}^2 \text{ or } A_{si}^2] & . \\ . & . & . \\ . & . & . \\ . & . & . \\ . & . & . \\ . & . & . \\ . & . & [A_{sc}^n \text{ or } A_{si}^n] \end{bmatrix} \end{bmatrix}$$

State Vector at Time k

The state vector of the buoys at time k is:

$$\begin{bmatrix} \begin{bmatrix} x_1^1(k) \\ x_2^1(k) \end{bmatrix} \\ \vdots \\ \begin{bmatrix} x_1^n(k) \\ x_2^n(k) \end{bmatrix} \end{bmatrix}$$

where the superscript indicates the buoy number and the subscript indicates the x_1 or x_2 status of the single buoy.

Input Vector

The input vector represents the external contributions of energy and water required by the buoys:

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

Matrix Input Vector

The matrix that multiplies the input vector takes into account the resources withdrawn:

$$\begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \\ \vdots \end{bmatrix}$$

specifically, the use of these matrix subblocks indicates the assignment of the vessel to a buoy, the request will therefore arrive at the buoy identified by a certain index i whose matrix subblock will have the 1's positioned on the diagonal.

Vector of Constant Terms

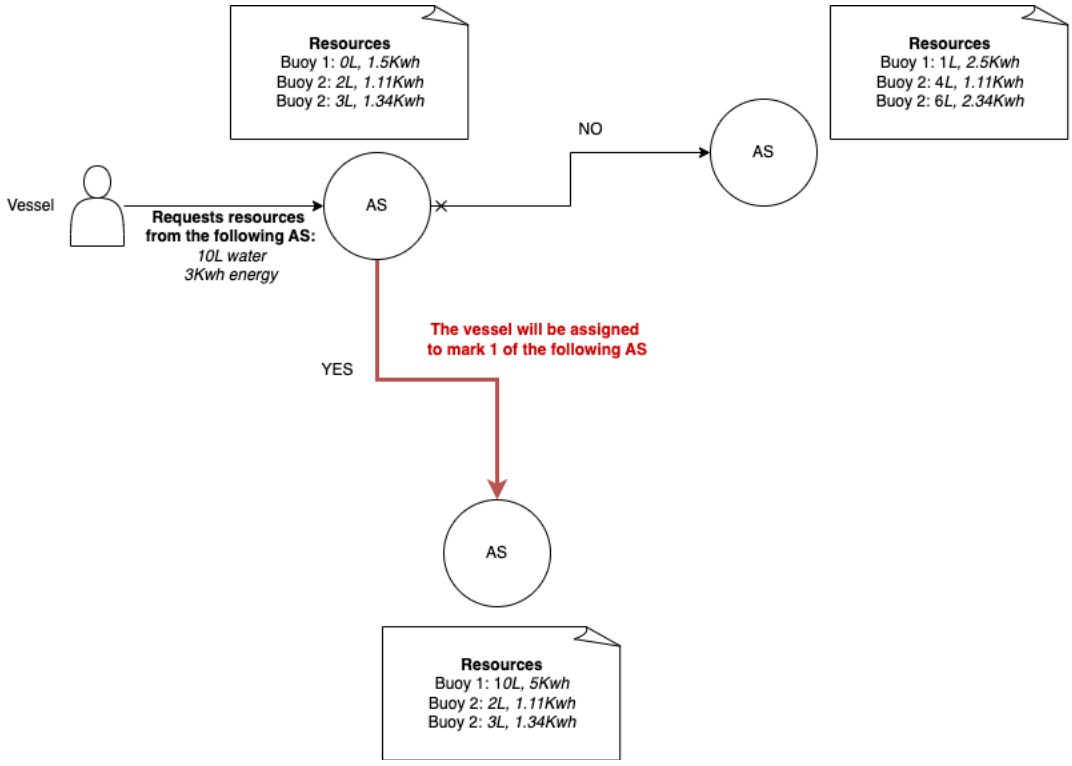
The vector of constant terms represents the fixed contributions of energy and water:

$$\begin{bmatrix} [C_{sc}^1 \text{ or } C_{si}^1] \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ [C_{sc}^n \text{ or } C_{si}^n] \end{bmatrix}$$

In summary, this matrix form represents the dynamic evolution of a system of buoys that accumulate energy and water, considering the internal dynamics of each buoy and the external contributions in terms of resources taken.

7 Request the distribution model between and within the ASes and assign requests

In the examples above we have addressed different request distribution patterns, each with advantages and disadvantages. The objective is to assign a request for a vessel to a buoy: all the cases that will be described will have the same condition in common, in which, if none of the buoys possess the requested resources, it will send the vessel to another AS. Below is an example that describes its behavior:

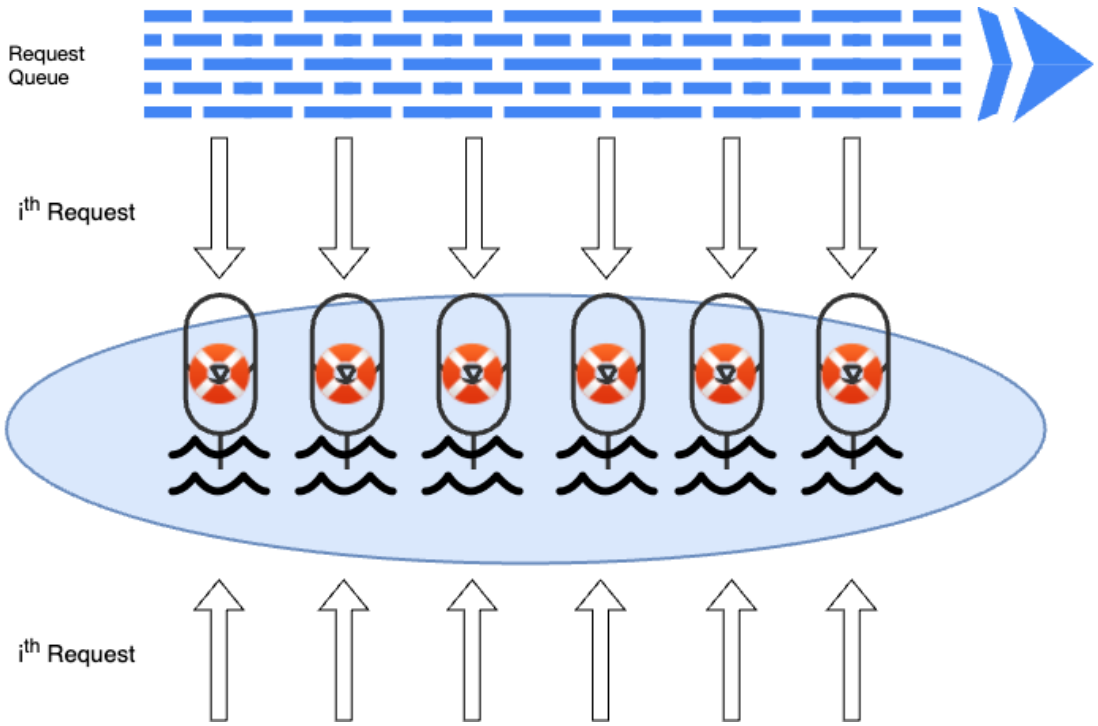


What changes in the following models will be related to the maximization of resource consumption at the buoy and the robustness of the data exchange which could be subject to sensor errors.
We therefore analyze the different cases and describe how they work.

7.1 Simple method

In this scenario we have a distribution done in the following way: the first buoy that can provide the resources is the one that will satisfy the request.

7.2 Round Robin method



In an iterative manner, the vessel is assigned in rotation to each buoy of the AS. If there is no buoy in the AS that can satisfy this request, it will send the vessel back to the first AS capable of satisfying it.

Description of the method in the context of origin

The Round Robin method is a scheduling algorithm used in operating systems to manage the execution of processes in a fair and balanced way. It is especially popular in time-sharing systems, where the goal is to provide a quick interactive experience to all users. Here is a detailed explanation of how it works:

Basic Principle

The requests will be taken care of and distributed in rotation among the buoys relating to a certain AS. Various cases may occur:

- a certain buoy of the AS cannot satisfy the request, in this case the request is forwarded to the next one which will perform the same check;
- no buoy within the AS can satisfy the request, in which case the vessel will be redirected to another AS which will have at least one buoy that can satisfy the request.

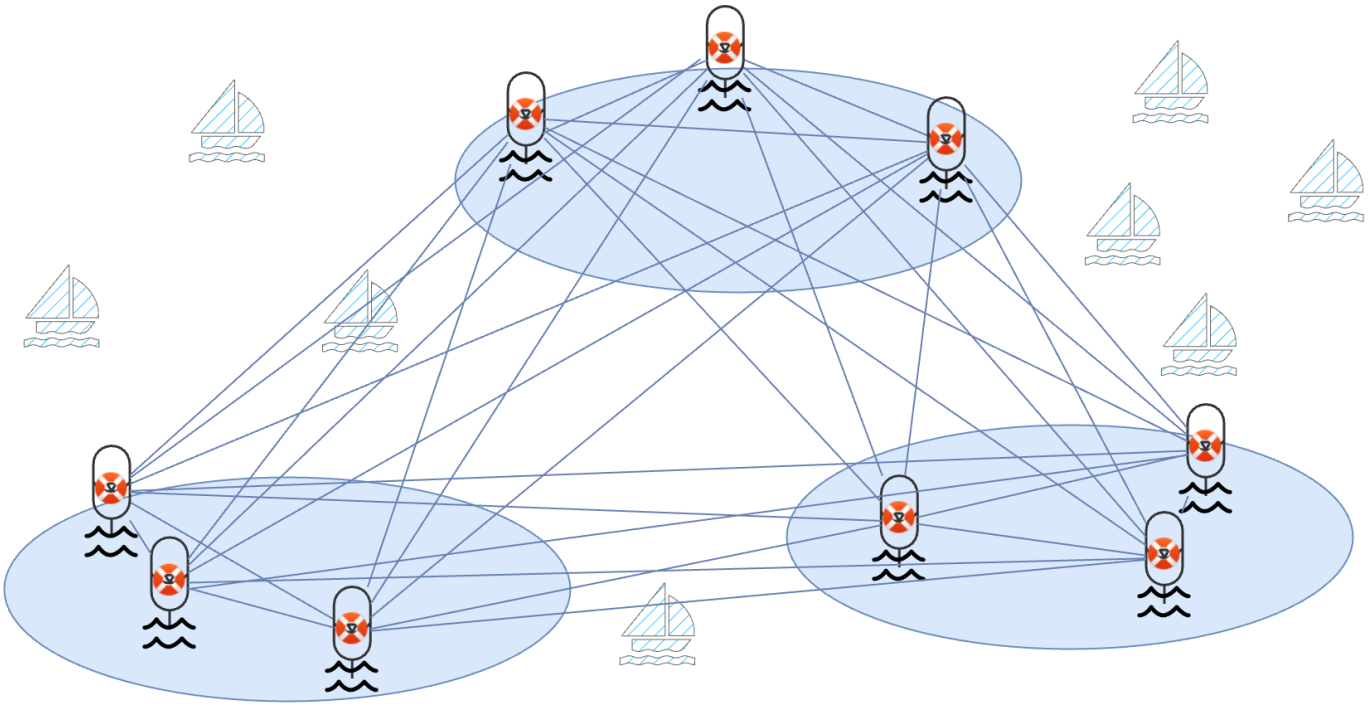
on each AS the load distribution on the buoys is performed in rotation on each of them, in this case the workload will be fair but the maximization of resource consumption at a buoy is not considered.

7.2.1 Function to assign vessel requests to buoys using round robin

Below is the code that allows you to perform a round robin assignment within each AS:

```
def assign_requests_round_robin(x1, x2, u1, u2, k, num_vessels, min_energy, max_energy, max_water, last_assigned_buoy):
    """
    Assigns requests to buoys in a round-robin fashion based on energy and water requirements.

    Parameters:
    x1 (2D array): Energy available at each buoy for each time step.
    x2 (2D array): Water available at each buoy for each time step.
    u1 (2D array): Energy requested by each vessel for each time step.
    u2 (2D array): Water requested by each vessel for each time step.
    k (int): Current time step.
    num_vessels (int): Number of vessels to assign.
    min_energy (float): Minimum energy threshold for a buoy.
    max_energy (float): Maximum energy threshold for a buoy.
    max_water (float): Maximum water threshold for a buoy.
    last_assigned_buoy (int): The index of the last buoy that was assigned a vessel.

    Returns:
    int: The index of the last buoy assigned.
    """

    # Iterate through each vessel
    for i in range(num_vessels):
        assigned = False  # Flag to check if the vessel has been assigned

        # Try to assign the vessel to a buoy in a round-robin manner
        for _ in range(len(x1)):  # Ensure we try each buoy exactly once
            j = (last_assigned_buoy + 1) % len(x1)  # Get the next buoy index in a circular manner

            # Check if the buoy can fulfill the energy and water request
            if (x1[j, k + 1] - u1[i, k] >= min_energy and x1[j, k + 1] - u1[i, k] <= max_energy and
                x2[j, k + 1] - u2[i, k] >= 0 and x2[j, k + 1] - u2[i, k] <= max_water):

                # Assign the request to buoy j
                x1[j, k + 1] -= u1[i, k]  # Deduct the energy requested by the vessel from the buoy
                x2[j, k + 1] -= u2[i, k]  # Deduct the water requested by the vessel from the buoy

                # Ensure buoy's energy does not fall below the minimum threshold
                if x1[j, k + 1] < min_energy:
                    x1[j, k + 1] = min_energy

                # Ensure buoy's water does not fall below zero
                if x2[j, k + 1] < 0:
                    x2[j, k + 1] = 0

                # Print the assignment details
                print(f"Vessel {i + 1} assigned to Buoy {j + 1}:")
                print(f"  Energy requested: {u1[i, k]}")
                print(f"  Water requested: {u2[i, k]}")
                print(f"  Updated energy for Buoy {j + 1}: {x1[j, k + 1]}")
                print(f"  Updated water for Buoy {j + 1}: {x2[j, k + 1]}")

                assigned = True  # Mark the vessel as assigned
                last_assigned_buoy = j  # Update the last assigned buoy index
                break  # Exit the loop as the vessel has been assigned

        # If the vessel could not be assigned to any buoy
        if not assigned:
            print(f"Vessel {i + 1} could not be assigned at time step {k + 1}")

    return last_assigned_buoy  # Return the index of the last assigned buoy
```

7.3 Greedy method



In the given context, the **Optimized Greedy Assignment** algorithm is applied to manage the allocation of energy and water resources from autonomous buoy systems to incoming vessels. Here's an explanation of how the algorithm fits into this application, along with its advantages and disadvantages.

Context of Application

Scenario

The system consists of multiple autonomous buoy systems, each with a set number of buoys. These buoys accumulate energy from renewable sources and store water. The buoys need to provide these resources to incoming vessels, which arrive at various times and have different energy and water demands.

Goals

1. **Efficient Resource Management**: Ensure that the buoys can effectively meet the demands of the vessels without depleting their resources below critical levels.
2. **Balanced Load Distribution**: Prevent any single buoy from being overburdened, thus ensuring the longevity and reliability of the buoy system.

Advantages of Optimized Greedy Assignment

1. **Simplicity and Ease of Implementation**:
    - The greedy algorithm is straightforward to understand and implement, making it suitable for real-time applications where quick decision-making is essential.
2. **Fair Resource Allocation**:
    - By choosing the buoy that can best handle a request with minimal impact, the algorithm helps distribute the load evenly across all buoys, preventing resource exhaustion in any single buoy.
3. **Scalability**:

- The algorithm can easily scale with the number of buoys and vessels. Even as the system grows, the greedy approach remains manageable in terms of computational complexity.

4. **Local Optimization**:
   - The algorithm makes optimal local decisions, which can often lead to good overall system performance, particularly in dynamic environments where conditions change frequently.
5. **Real-Time Response**:
   - The greedy nature allows for quick decisions, which is critical in real-time systems where delays in allocation could lead to resource shortages or other operational issues.

Disadvantages of Optimized Greedy Assignment

1. **Global Optimality**:
   - The algorithm focuses on local optimization and does not guarantee a globally optimal solution. This means that while each individual decision is optimal, the overall resource allocation might not be the best possible.
2. **Sensitivity to Parameters**:
   - The effectiveness of the algorithm can be sensitive to the chosen parameters, such as the minimum energy threshold and the rate of energy and water production. Incorrect parameter settings can lead to suboptimal performance.
3. **Resource Depletion Risk**:
   - If the arrival rate of vessels or their resource demands are higher than expected, the buoys might deplete their resources faster than they can replenish, leading to potential service failures.
4. **Computational Overhead**:
   - Although generally efficient, the algorithm requires checking the impact on all buoys for each vessel request, which can become computationally intensive with a large number of buoys and requests.
5. **Lack of Long-Term Planning**:
   - The greedy approach does not consider future requests or changes in the system. This lack of foresight can lead to situations where resources are allocated inefficiently over the long term.

Summary

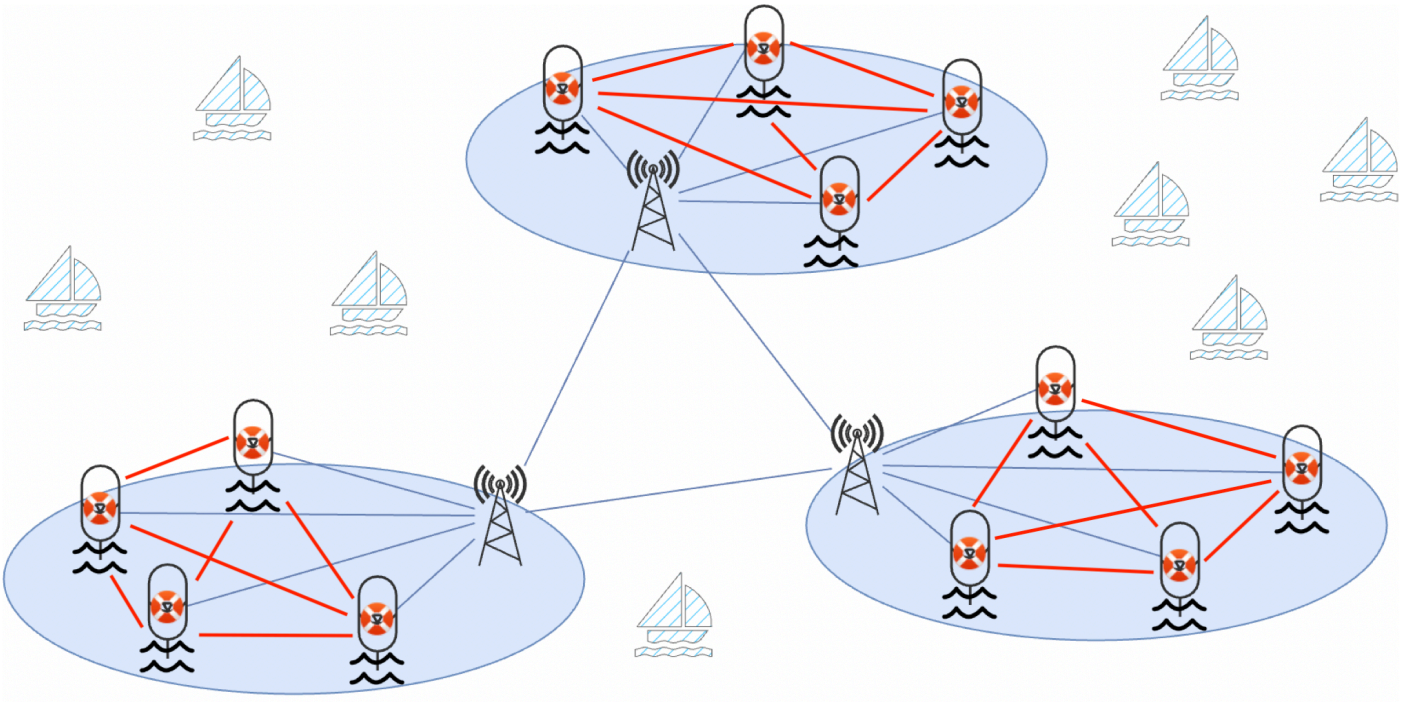
The **Optimized Greedy Assignment** algorithm provides a practical solution for managing resources in autonomous buoy systems, balancing simplicity and efficiency with fair resource allocation. However, its focus on local optimization and sensitivity to parameters mean that it may not always produce the best global results. In dynamic and real-time applications, it offers quick and often effective decision-making, but it requires careful tuning and may need supplementary strategies to handle extreme scenarios or long-term planning.

7.3.1 Function to assign vessel requests to buoy systems using an optimized greedy algorithm

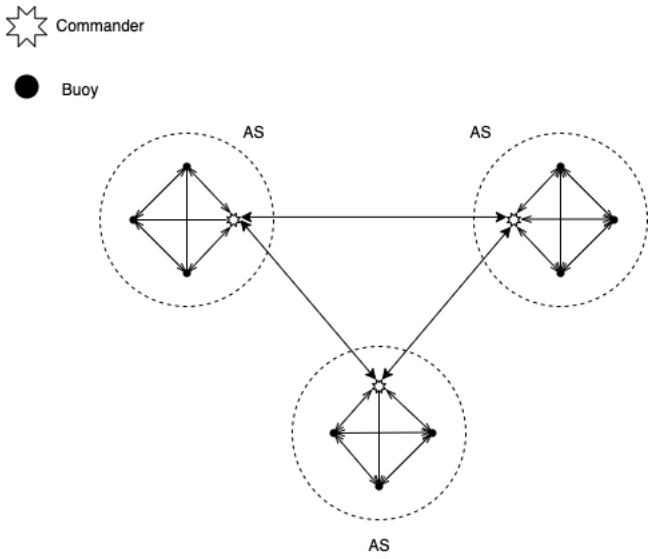
Below is the code used to assign resources according to a greedy algorithm optimized within each AS:

```
def assign_requests_optimized(system, system_index, u1, u2, k, num_vessels, min_energy, max_energy, max_water):
    """
    Optimally assigns requests to buoys based on energy and water requirements, choosing the buoy with the minimum impact.

    Parameters:
    system (tuple of 2D arrays): Contains energy (x1) and water (x2) availability at each buoy for each time step.
    system_index (int): Index of the current system being processed.
    u1 (2D array): Energy requested by each vessel for each time step.
    u2 (2D array): Water requested by each vessel for each time step.
    k (int): Current time step.
    num_vessels (int): Number of vessels to assign.
    min_energy (float): Minimum energy threshold for a buoy.
    max_energy (float): Maximum energy threshold for a buoy.
    max_water (float): Maximum water threshold for a buoy.

    Returns:
    None
    """
    x1, x2 = system  # Unpack the energy and water arrays from the system tuple

    # Iterate through each vessel
    for i in range(num_vessels):
        best_buoy = None  # Initialize the best buoy index
        min_impact = float('inf')  # Initialize the minimum impact value to infinity

        # Iterate through each buoy to find the best one to fulfill the request
        for j in range(len(x1)):
            # Check if the buoy can fulfill the energy and water request
            if (x1[j, k + 1] - u1[i, k] >= min_energy and x2[j, k + 1] - u2[i, k] >= 0):
                # Calculate the impact of assigning this buoy
                impact = (x1[j, k + 1] - u1[i, k]) + (x2[j, k + 1] - u2[i, k])

                # Update the best buoy if this impact is less than the current minimum impact
                if impact < min_impact:
                    min_impact = impact
                    best_buoy = j

        # If a suitable buoy was found, assign the request to it
        if best_buoy is not None:
            x1[best_buoy, k + 1] -= u1[i, k]  # Deduct the energy requested by the vessel from the buoy
            x2[best_buoy, k + 1] -= u2[i, k]  # Deduct the water requested by the vessel from the buoy

            # Ensure buoy's energy does not fall below the minimum threshold
            if x1[best_buoy, k + 1] < min_energy:
                x1[best_buoy, k + 1] = min_energy

            # Ensure buoy's water does not fall below zero
            if x2[best_buoy, k + 1] < 0:
                x2[best_buoy, k + 1] = 0

            # Print the assignment details
            print(f"Vessel {i + 1} assigned to Buoy {best_buoy + 1} in System {system_index + 1}:")
            print(f"  Energy requested: {u1[i, k]}")
            print(f"  Water requested: {u2[i, k]}")
            print(f"  Updated energy for Buoy {best_buoy + 1} in System {system_index + 1}: {x1[best_buoy, k + 1]}")
            print(f"  Updated water for Buoy {best_buoy + 1} in System {system_index + 1}: {x2[best_buoy, k + 1]}")
        else:
            # Print a message if the vessel could not be assigned to any buoy
            print(f"Vessel {i + 1} could not be assigned at time step {k + 1}")
```

7.4 Problem of the Byzantine generals



The condensed digraph resulting from the exchange of information can be considered mutable at different moments of time. This problem, which concerns reaching a consensus among multiple (general) participants in the presence of components that can act maliciously or fail, involves continuous communication between the (general) nodes of the network. In particular, in practical scenarios, the communication channels between nodes may not always be stable and reliable, and the topology of the communication network graph may vary over time due to various factors such as temporary node failures, changes in network connectivity, network, interference, etc. Therefore, the communication graph may be different at different instants of time. This mutability of the graph can make reaching consensus more complex, since algorithms must be designed to handle not only the presence of malicious nodes but also variations in the topology of the network. Below is the generic graph whose orientation, for the purposes of consensus in an AS sink, varies with the assignment of the vessel to a buoy (this implies that the arrows would become unidirectional whose nodes represent the strongly connected components).



In our context the problem of the Byzantine generals has been used in two different contexts: between different ASs and within each AS. Each commander who may be a transmitter or a special buoy will be responsible for making decisions regarding the assignment of a vessel to a buoy or another AS. To choose the best possible version of the algorithm, it would be desirable to find information about transmission protocols, types of transmission, sources of interference, etc. Surely the normal version of the algorithm (which requires a number of commanders n equal to 3t+1 with t equal to the number of traitors is not the best possible solution, since the high availability of the data must also be guaranteed if a commander buoy is compromised and start sending incorrect data).

Description of the Byzantine Generals Problem

The Byzantine Generals Problem is a classic problem of distributed consensus first formulated by Leslie Lamport, Robert Shostak, and Marshall Pease in 1982. The problem is a metaphor for describing how distributed systems can reach a correct agreement even in the presence of malfunctioning or malicious nodes.

Problem Scenario

Imagine a group of Byzantine army generals who need to agree on a common strategy, either to attack or to retreat. The generals are in separate camps and can only communicate through messengers. Some of these generals might be traitors and try to confuse the others by sending conflicting information.

The objective is for all loyal generals to reach a consensus on the same decision (attack or retreat) and ensure that the traitors cannot cause the loyal generals to make different decisions.

Assumptions of the Problem

1. **Generals and Messengers**:
    - Each general can send messages to the other generals.
    - Messages can be intercepted, delayed, or falsified by the traitors.
2. **Loyalty and Betrayal**:
    - Some generals might be traitors and send false information.
    - Loyal generals must still reach a common agreement.

Solution to the Problem

The problem can be solved with distributed consensus protocols that ensure:

1. **Agreement**: All loyal generals agree on the same decision.
2. **Validity**: If all loyal generals propose the same decision, then that decision will be the final outcome.

Lamport, Shostak, and Pease's Algorithm

A well-known algorithm to solve this problem includes the following steps:

1. **Consensus Initiation**: Each general sends their proposal to all other generals.
2. **Message Collection**: Each general collects the proposals received from other generals.
3. **Message Relaying**: Each general forwards the information received to other generals for a sufficient number of rounds to ensure consensus.

4. **Final Decision**: Each general uses a majority function (or another aggregation function) to determine the final decision based on the messages received.

Application of the Byzantine Generals Problem in Autonomous Buoy Systems

In the simulation of autonomous buoys, the Byzantine Generals Problem is used to ensure that decisions regarding the distribution of resources (energy and water) are correct and consistent, even in the presence of erroneous or malicious information from some systems.

Operation in the Context of Autonomous Buoy Systems

1. **Information Gathering**:
   - Each Autonomous System (AS) gathers information about resources (energy and water) from the buoys of other AS.
2. **Information Verification**:
   - Each AS verifies the information received using a majority mechanism. This helps to filter out erroneous or malicious information.
3. **Consensus**:
   - Decisions about resources are made based on the verified information, ensuring that even if some AS are compromised, the final decision will be correct and based on the majority of reliable information.

Advantages and Disadvantages

Advantages:

- **Robustness**: Protects the system against failures and malicious attacks.
- **Consistency**: Ensures that all AS have a consistent view of the state of resources.
- **Reliability**: Increases the overall reliability of the system in dynamic and uncertain environments.
- **Scalability**: Can be applied to large distributed networks.

Disadvantages:

- **Complexity**: Increases the system's complexity, requiring sophisticated algorithms for consensus.
- **Delays**: Introduces delays in decision-making due to the need to gather and verify data.
- **Communication Overhead**: Requires substantial information exchange between AS, increasing network traffic.
- **Computational Costs**: Requires additional computational resources for processing and verifying data.

Conclusion

The use of the Byzantine Generals Problem in the context of autonomous buoys provides a robust and reliable solution to ensure correct resource distribution decisions, even in the presence of erroneous or malicious information. However, this robustness comes at the cost of increased complexity, delays, and communication overhead. The decision to implement such a mechanism depends on the importance of resilience and data consistency compared to operational costs.

7.4.1 Function to share and verify data between systems thanks to the Byzantine generals problem

Below is the code used for verifying and sharing data between ASes and within each of them in order to reach consensus on the assignment of the vessel's request to a buoy:

```
def assign_requests_to_buoys(system, u1, u2, k, num_vessels, min_energy, max_energy, max_water, system_index):
    """
    Assigns requests to buoys based on energy and water requirements, choosing the buoy with the minimum impact.

    Parameters:
    system (tuple of 2D arrays): Contains energy (x1) and water (x2) availability at each buoy for each time step.
    u1 (2D array): Energy requested by each vessel for each time step.
    u2 (2D array): Water requested by each vessel for each time step.
    k (int): Current time step.
    num_vessels (int): Number of vessels to assign.
    min_energy (float): Minimum energy threshold for a buoy.
    max_energy (float): Maximum energy threshold for a buoy.
    max_water (float): Maximum water threshold for a buoy.
    system_index (int): Index of the current system being processed.

    Returns:
    None
    """
    x1, x2 = system  # Unpack the energy and water arrays from the system tuple

    # Iterate through each vessel
    for i in range(num_vessels):
        best_buoy = None  # Initialize the best buoy index
        min_impact = float('inf')  # Initialize the minimum impact value to infinity

        # Iterate through each buoy to find the best one to fulfill the request
        for j in range(len(x1)):
            # Check if the buoy can fulfill the energy and water request
            if (x1[j, k + 1] - u1[i, k] >= min_energy and x2[j, k + 1] - u2[i, k] >= 0):
                # Calculate the impact of assigning this buoy
                impact = (x1[j, k + 1] - u1[i, k]) + (x2[j, k + 1] - u2[i, k])

                # Update the best buoy if this impact is less than the current minimum impact
                if impact < min_impact:
                    min_impact = impact
                    best_buoy = j

        # If a suitable buoy was found, assign the request to it
        if best_buoy is not None:
            x1[best_buoy, k + 1] -= u1[i, k]  # Deduct the energy requested by the vessel from the buoy
            x2[best_buoy, k + 1] -= u2[i, k]  # Deduct the water requested by the vessel from the buoy

            # Ensure buoy's energy does not fall below the minimum threshold
            if x1[best_buoy, k + 1] < min_energy:
                x1[best_buoy, k + 1] = min_energy

            # Ensure buoy's water does not fall below zero
            if x2[best_buoy, k + 1] < 0:
                x2[best_buoy, k + 1] = 0

            # Print the assignment details
            print(f"Vessel {i + 1} assigned to Buoy {best_buoy + 1} in System {system_index + 1}:")
            print(f"  Energy requested: {u1[i, k]}")
            print(f"  Water requested: {u2[i, k]}")
            print(f"  Updated energy for Buoy {best_buoy + 1} in System {system_index + 1}: {x1[best_buoy, k + 1]}")
            print(f"  Updated water for Buoy {best_buoy + 1} in System {system_index + 1}: {x2[best_buoy, k + 1]}")
        else:
            # Print a message if the vessel could not be assigned to any buoy
            print(f"Vessel {i + 1} could not be assigned to any Buoy in System {system_index + 1}")

def assign_requests_to_systems(systems, u1, u2, k, num_vessels, min_energy, max_energy, max_water):
    """
    Assigns requests to buoys across multiple systems.

    Parameters:
    systems (list of tuples): List of systems, each containing energy (x1) and water (x2) availability at each buoy for each time step.
    u1 (2D array): Energy requested by each vessel for each time step.
    u2 (2D array): Water requested by each vessel for each time step.
    k (int): Current time step.
    num_vessels (list of int): Number of vessels to assign at each time step.
    min_energy (float): Minimum energy threshold for a buoy.
    max_energy (float): Maximum energy threshold for a buoy.
    max_water (float): Maximum water threshold for a buoy.

    Returns:
    None
    """
    # Iterate through each vessel to be assigned at the current time step
    for i in range(num_vessels[k]):
        assigned = False  # Flag to check if the vessel has been assigned

        # Iterate through each system to find a suitable buoy
        for system_index, system in enumerate(systems):
            x1, x2 = system  # Unpack the energy and water arrays from the system tuple

            # Iterate through each buoy in the current system
            for j in range(len(x1)):
                # Check if the buoy can fulfill the energy and water request
                if (x1[j, k + 1] - u1[i, k] >= min_energy and x2[j, k + 1] - u2[i, k] >= 0):
                    print(f"Commander of System {system_index + 1} is considering assignment of Vessel {i + 1} to Buoy {j + 1}")

                    # Attempt to assign the request to the best buoy in the current system
                    assign_requests_to_buoys(system, u1, u2, k, num_vessels[k], min_energy, max_energy, max_water, system_index)
                    assigned = True  # Mark the vessel as assigned
                    break  # Exit the buoy loop as the vessel has been assigned

            if assigned:
                break  # Exit the system loop as the vessel has been assigned

        # If the vessel could not be assigned to any buoy in any system
        if not assigned:
            print(f"Vessel {i + 1} could not be assigned to any System at time step {k + 1}")

def verify_and_consensus(systems, k):
    """
    Performs verification and consensus across multiple systems to ensure data consistency.

    Parameters:
    systems (list of tuples): List of systems, each containing energy (x1) and water (x2) availability at each buoy for each time step.
    k (int): Current time step.

    Returns:
    None
    """
    print(f"Consensus process at time step {k + 1}")

    # Iterate through each system to perform verification and consensus
    for system_index, system in enumerate(systems):
        x1, x2 = system  # Unpack the energy and water arrays from the system tuple

        # Initialize arrays to collect reports from other systems
        energy_reports = np.zeros((len(systems), len(x1)))
        water_reports = np.zeros((len(systems), len(x1)))

        # Collect energy and water reports from other systems
        for other_system_index, other_system in enumerate(systems):
            if other_system_index != system_index:
                other_x1, other_x2 = other_system  # Unpack the energy and water arrays from the other system tuple

                # Collect reports from each buoy in the other system
                for i in range(len(x1)):
                    energy_reports[other_system_index, i] = other_x1[i, k + 1]
                    water_reports[other_system_index, i] = other_x2[i, k + 1]

        # Initialize arrays to hold the verified energy and water values
        verified_energy = np.zeros(len(x1))
        verified_water = np.zeros(len(x1))

        # Determine the most common energy and water values for each buoy
        for i in range(len(x1)):
            energy_counts = Counter(energy_reports[:, i])
            water_counts = Counter(water_reports[:, i])
            verified_energy[i] = energy_counts.most_common(1)[0][0]
            verified_water[i] = water_counts.most_common(1)[0][0]

        # Update the system with the verified energy and water values
        for i in range(len(x1)):
            print(f"System {system_index + 1} updating verified data for Buoy {i + 1}:")
            print(f"  Verified energy: {verified_energy[i]}")
            print(f"  Verified water: {verified_water[i]}")
            x1[i, k + 1] = verified_energy[i]
            x2[i, k + 1] = verified_water[i]
```

8 Code implementation and results

In this section we will discuss the different simulations implemented in the project by inserting the results obtained and the related code (available at the following link: https://github.com/dany27041991/ACT/blob/main/PROJECT/Buoy%20network%20project.ipynb).

PLEASE NOTE: In the following simulations we have considered the stochastic arrival of n requests from vessels with an interval of one hour. For the data considered we estimated an overproduction of resources and therefore decided to consider the most important vessel flow during the day. If necessary, the code can be easily adapted to its construction. Below is the arrival logic of requests from vessels.

In implementing the code we opted for different scenarios, listed below:

1. Implementation of three autonomous systems each consisting of 3 buoys with random generation of vessel requests for time slots, specifically:
    a. 02:00am – 05:00am: zero installments (0)
    b. 06:00am - 10:00am: low rate (1 to max_vessels / 3)
    c. 10:00am -18:00pm: high rate (1 to max_vessels)
    d. 18:00pm - 22:00pm: low rate (1 to max_vessels / 3)
    e. 10:00pm - 06:00am: rate almost zero (0 to 1)

    the assignment takes place at the first mark with available resources.
2. Implementation of three autonomous systems each consisting of 3 buoys with random generation of vessel requests for time slots, specifically:
    a. 02:00am – 05:00am: zero installments (0)
    b. 06:00am - 10:00am: low rate (1 to max_vessels / 3)
    c. 10:00am -18:00pm: high rate (1 to max_vessels)
    d. 18:00pm - 22:00pm: low rate (1 to max_vessels / 3)
    e. 10:00pm - 06:00am: rate almost zero (0 to 1)

    the assignment occurs according to a round robin criterion even between autonomous systems.
3. Implementation of three autonomous systems each consisting of 3 buoys with random generation of vessel requests for time slots, specifically:
    a. 02:00am – 05:00am: zero installments (0)
    b. 06:00am - 10:00am: low rate (1 to max_vessels / 3)
    c. 10:00am -18:00pm: high rate (1 to max_vessels)
    d. 18:00pm - 22:00pm: low rate (1 to max_vessels / 3)
    e. 10:00pm - 06:00am: rate almost zero (0 to 1)

    The assignment takes place according to the following method:
    - Data Collection: Each buoy collects its own status data (available energy and water).
    - Data Sharing: Buoys share their status data with other buoys in the system.
    - Data Verification: Each buoy verifies the data received from the other buoys by comparing them with its own data and with the data received from other buoys.
    - Data Consensus: Buoys reach a consensus on the correct data through a consensus protocol such as the Byzantine Generals Protocol.

    This mode manages the possibility that some buoys may provide incorrect values to every other buoy thanks to a completely distributed communication throughout the entire network of buoys, and also makes resource consumption efficient by maximizing it on each individual buoy.

4. It is an evolved version of problem 5 which exploits the problem of the Byzantine generals for the exchange of information between different autonomous systems, in this mode there will be a commander for each autonomous system who will then direct within his own autonomous system or will redirect to another.

For brevity we will only analyze complete cases with multiple ASs, specifically 1, 2, 3 and 4.

8.1 Case n°1

See section 1.2 of the code at the previous link.

Global Parameters
The initial parameters are defined, including:

- The number of buoy systems (`num_autonomous_systems = 3`).
- The number of buoys per system (`n = 3`).
- The constant energy production (`cr = 4.6` units per timestep).
- The energy-water conversion parameter (`kc = 0.05`).
- The simulation duration (`timesteps = 24`).
- Initial levels of energy and water (`initial_energy = 5` and `initial_water = 10`).
- Maximum levels of energy and water (`max_energy = 5` and `max_water = 10`).
- Minimum energy threshold (`min_energy = 1`).
- Target water level (`x2_target = 10`).
- Maximum number of vessels per timestep (`max_vessels = 10`).

State Vector Initialization
The `initialize_state_vectors` function creates and initializes the state vectors for energy and water for each buoy with the updated initial conditions.

Vessel Number Generation
The `generate_num_vessels_per_timestep` function generates the number of vessels for each timestep, following a random distribution with varying rates of vessel arrivals.

Random Request Generation
The `generate_random_requests` function creates random requests for energy and water from the vessels arriving at each timestep.

Buoy State Update
The `update_buoys` function updates the energy and water levels of the buoys based on production and energy-water conversion. It ensures that the levels remain within the defined maximum thresholds.

Request Assignment
The `assign_requests_to_systems` function assigns vessel requests to the buoys, updating their energy and water levels to balance the load across all buoys.

Simulation Execution
The `run_simulation_multiple_systems` function runs the simulation for the specified number of timesteps, updating buoy states and assigning vessel requests at each timestep.

Results Visualization
The `plot_results_multiple_systems` function visualizes the simulation results, displaying the energy and water levels for each buoy over time. It includes labels for each timestep, formatted as hours of the day.

Summary of Results
Energy Levels

- The energy levels of all buoys in each system show significant fluctuations throughout the simulation.
- The energy levels start at the maximum threshold (5 units) and fluctuate due to the varying demands from the vessels.
- The continuous rise and fall in energy levels suggest a balance between energy production and consumption by the vessels.

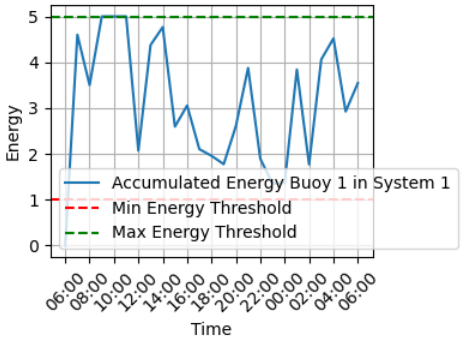
Water Levels

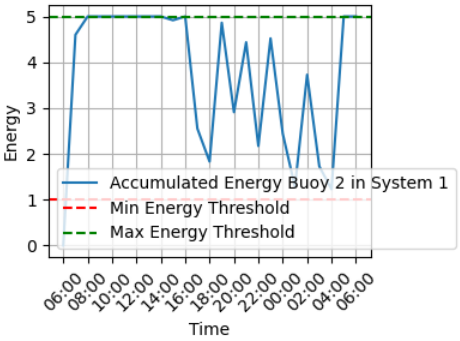
- The water levels exhibit considerable variations, starting at the maximum threshold (10 units).
- There are instances where the water levels drop to lower values, indicating high water usage by the vessels and efficient replenishment cycles.

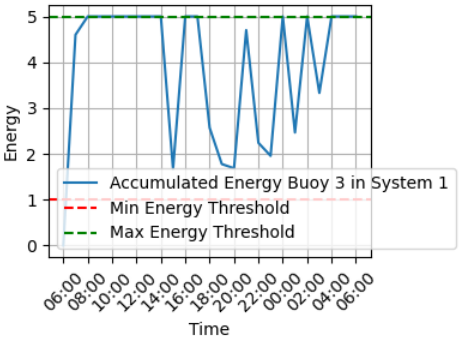
Request Assignment

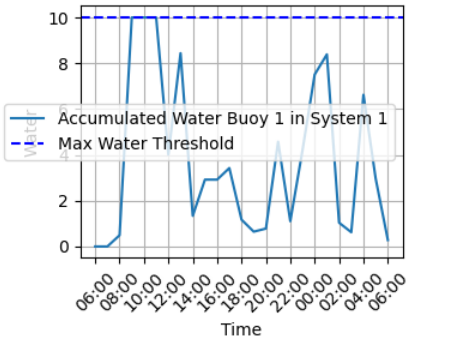
- The assignment method ensures balanced utilization of all buoys, preventing any single buoy from being overloaded.
- The method effectively assigns requests while maintaining energy and water levels within defined thresholds, ensuring reliable operation.

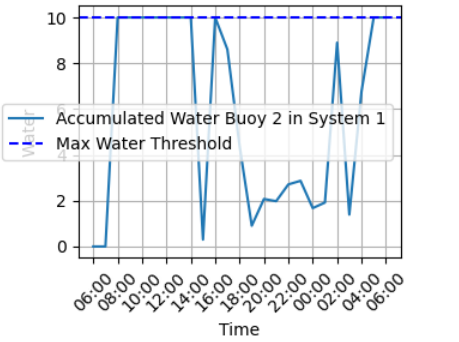
8.1.1 Results

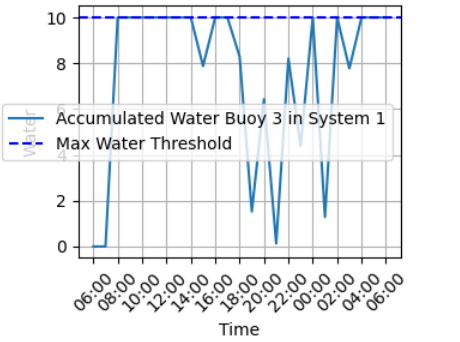

Energy
Time
Accumulated Energy Buoy 1 in System 1
Min Energy Threshold
Max Energy Threshold

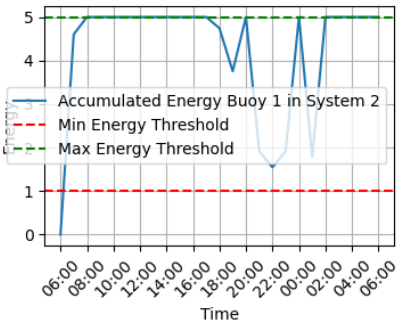

Energy
Time
Accumulated Energy Buoy 2 in System 1
Min Energy Threshold
Max Energy Threshold

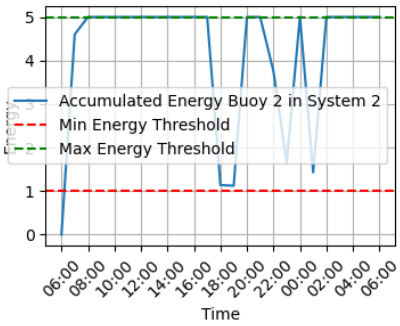

Energy
Time
Accumulated Energy Buoy 3 in System 1
Min Energy Threshold
Max Energy Threshold

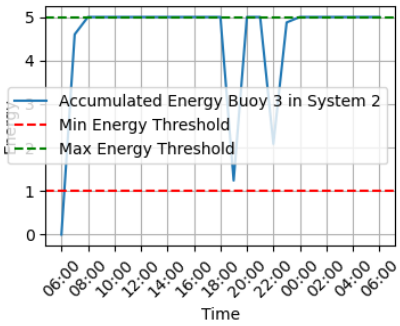

Water
Time
Accumulated Water Buoy 1 in System 1
Max Water Threshold

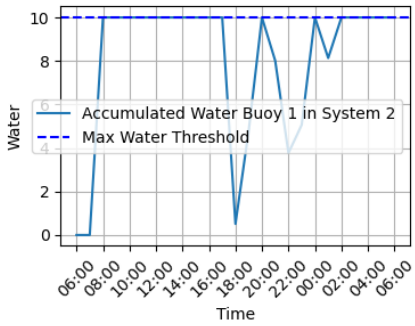

Water
Time
Accumulated Water Buoy 2 in System 1
Max Water Threshold

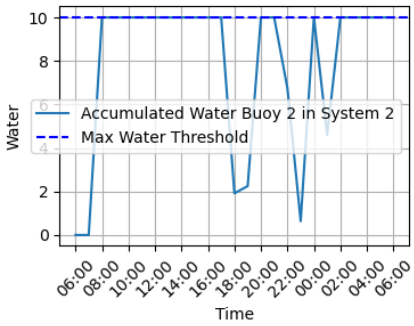

Water
Time
Accumulated Water Buoy 3 in System 1
Max Water Threshold

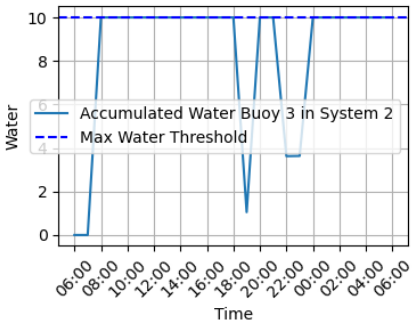

Energy
Time
Accumulated Energy Buoy 1 in System 2
Min Energy Threshold
Max Energy Threshold

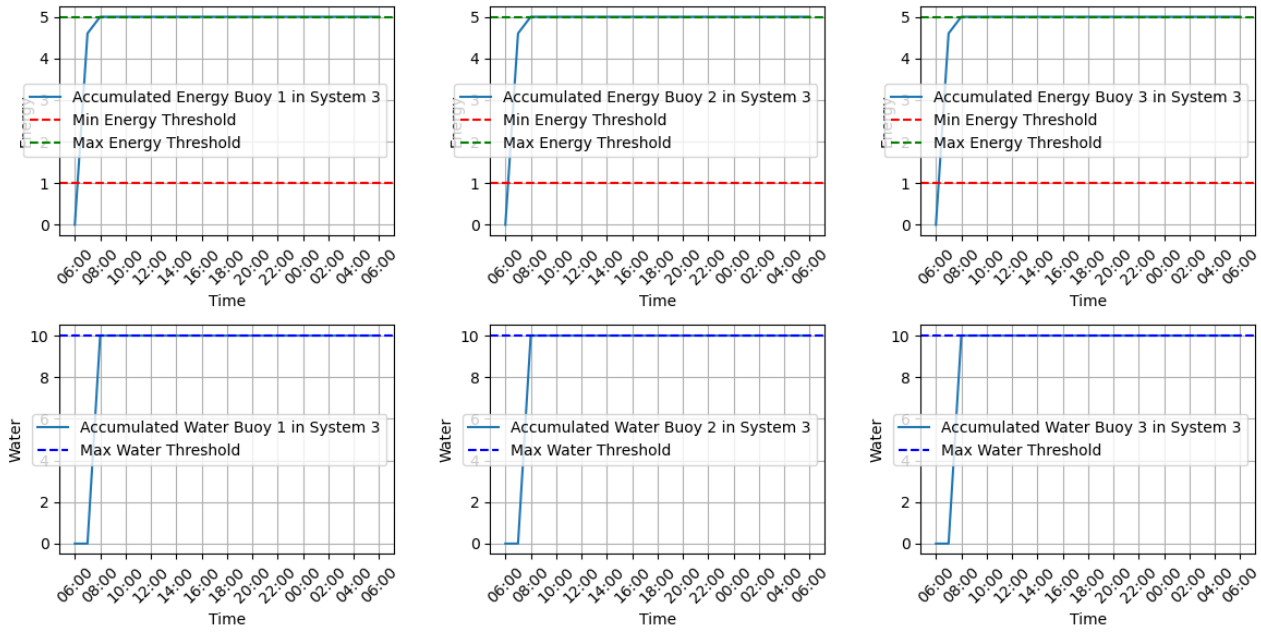

Energy
Time
Accumulated Energy Buoy 2 in System 2
Min Energy Threshold
Max Energy Threshold


Energy
Time
Accumulated Energy Buoy 3 in System 2
Min Energy Threshold
Max Energy Threshold


Water
Time
Accumulated Water Buoy 1 in System 2
Max Water Threshold


Water
Time
Accumulated Water Buoy 2 in System 2
Max Water Threshold


Water
Time
Accumulated Water Buoy 3 in System 2
Max Water Threshold



The provided graphs display the accumulated energy and water levels for each buoy in three autonomous buoy systems over a 24-hour simulation period. Below is an analysis of the results for each system.

System 1

Energy Levels:

- **Buoy 1**: The energy levels fluctuate significantly throughout the simulation. The energy often reaches the maximum threshold of 5 units and occasionally approaches the minimum threshold of 1 unit, indicating high energy demand and usage.
- **Buoy 2**: The energy levels start at the maximum and frequently drop close to the minimum threshold, showing similar usage patterns as Buoy 1.
- **Buoy 3**: The energy levels also exhibit significant fluctuations, reaching the maximum threshold multiple times, suggesting effective energy management and high energy consumption.

Water Levels:

- **Buoy 1**: The water levels fluctuate, often nearing the maximum threshold of 10 units but also showing significant drops, indicating frequent water usage and replenishment.
- **Buoy 2**: The water levels exhibit frequent fluctuations, similar to Buoy 1, suggesting consistent water usage by vessels.
- **Buoy 3**: The water levels show regular variations, often dropping to lower values, indicating efficient water management and high demand.

System 2

Energy Levels:

- **Buoy 1**: The energy levels start at the maximum threshold and show significant fluctuations, similar to System 1, indicating high energy demand and usage.

- **Buoy 2**: The energy levels frequently reach the maximum threshold and drop close to the minimum threshold, showing high usage.
- **Buoy 3**: The energy levels also exhibit significant fluctuations, similar to the other buoys, indicating balanced energy production and consumption.

Water Levels:

- **Buoy 1**: The water levels show considerable variation, often reaching the maximum threshold and dropping to lower values, indicating frequent usage.
- **Buoy 2**: The water levels exhibit frequent drops and replenishment cycles, similar to Buoy 1.
- **Buoy 3**: The water levels show regular variations, indicating consistent water usage and replenishment.

System 3

Energy Levels:

- **Buoy 1**: The energy levels quickly reach the maximum threshold and remain there, indicating no significant energy usage.
- **Buoy 2**: Similar to Buoy 1, the energy levels quickly reach and stay at the maximum threshold, indicating a lack of energy demand.
- **Buoy 3**: The energy levels also quickly reach the maximum threshold and remain constant, suggesting no significant energy consumption.

Water Levels:

- **Buoy 1**: The water levels quickly reach the maximum threshold and stay there, indicating no significant water usage.
- **Buoy 2**: Similar to Buoy 1, the water levels quickly reach and stay at the maximum threshold.
- **Buoy 3**: The water levels also quickly reach and remain at the maximum threshold, suggesting no significant water demand.

4.1.2 General Observations

1. **Fluctuation Patterns**: Systems 1 and 2 show similar patterns of significant fluctuations in energy and water levels, indicating high demand and usage by vessels. System 3, however, shows little to no fluctuation, suggesting no significant demand for energy or water.
2. **Threshold Compliance**: The energy and water levels generally remain within the defined thresholds, ensuring that the buoys do not exceed their storage capacities.
3. **Demand-Driven Consumption**: The frequent drops in both energy and water levels in Systems 1 and 2 suggest a high and regular demand from the vessels, requiring the buoys to continuously supply resources. System 3's constant levels suggest a lack of demand.

These results reflect the efficiency and responsiveness of the autonomous buoy systems in managing energy and water resources under varying demand conditions, with Systems 1 and 2 experiencing high usage and System 3 experiencing minimal usage.

8.2 Case n°2

See section 2.2 of the code at the previous link.

Global Parameters
The initial parameters are defined, including:

- The number of buoy systems (`num_autonomous_systems = 3`).
- The number of buoys per system (`n = 3`).
- The constant energy production (`cr = 4.6` units per timestep).
- The energy-water conversion parameter (`kc = 0.05`).
- The simulation duration (`timesteps = 24`).
- Initial levels of energy and water (`initial_energy = 0` and `initial_water = 0`).
- Maximum levels of energy and water (`max_energy = 5` and `max_water = 10`).
- Minimum energy threshold (`min_energy = 1`).
- Target water level (`x2_target = 10`).
- Maximum number of vessels per timestep (`max_vessels = 10`).

State Vector Initialization
The `initialize_state_vectors` function creates and initializes the state vectors for energy and water for each buoy.

Vessel Number Generation
The `generate_num_vessels_per_timestep` function generates the number of vessels for each timestep, following a random distribution with varying rates of vessel arrivals.

Random Request Generation
The `generate_random_requests` function creates random requests for energy and water from the vessels arriving at each timestep.

Buoy State Update
The `update_buoys` function updates the energy and water levels of the buoys based on production and energy-water conversion. It ensures that the levels remain within the defined maximum thresholds.

Request Assignment
The `assign_requests_round_robin` function assigns vessel requests to the buoys using a round-robin method, balancing the load across all buoys and ensuring that energy and water levels are updated correctly.

Simulation Execution
The `run_simulation_round_robin` function runs the simulation for the specified number of timesteps, updating buoy states and assigning vessel requests at each timestep using the round-robin method.

Results Visualization
The `plot_results_multiple_systems` function visualizes the simulation results, displaying the energy and water levels for each buoy over time.

Summary of Results

Energy Levels

- The energy levels of all buoys in each system show significant fluctuations throughout the simulation.

- The energy levels often reach the maximum threshold (5 units) and occasionally approach the minimum threshold (1 unit), indicating efficient energy management and high demand.
- The continuous rise and fall in energy levels suggest a balance between energy production and consumption by the vessels.

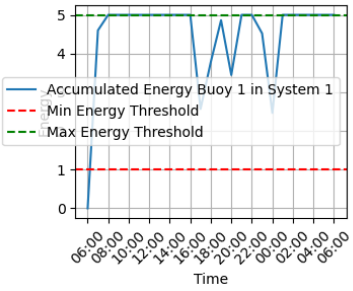
Water Levels

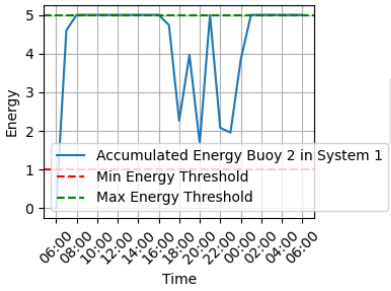
- The water levels exhibit considerable variations, frequently reaching the maximum threshold (10 units).
- There are multiple instances where the water levels drop to zero, indicating high water usage by the vessels and efficient replenishment cycles.

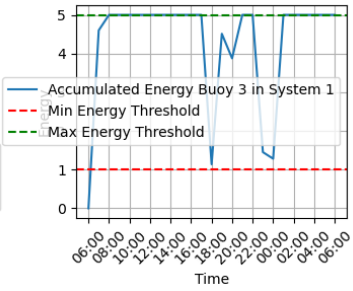
Round Robin Assignment

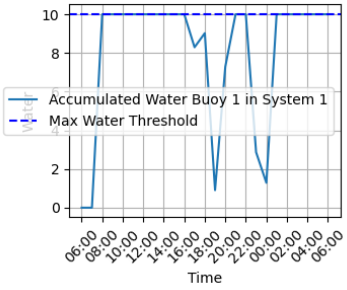
- The round-robin method ensures balanced utilization of all buoys, preventing any single buoy from being overloaded.
- The method effectively assigns requests while maintaining energy and water levels within defined thresholds, ensuring reliable operation.

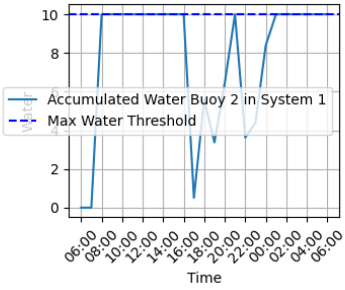
8.2.1 Results

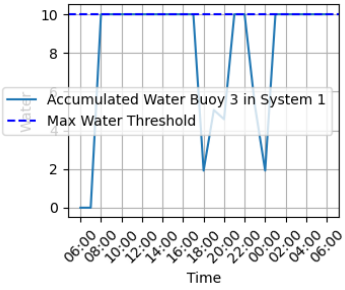

Accumulated Energy Buoy 1 in System 1
Min Energy Threshold
Max Energy Threshold
Time

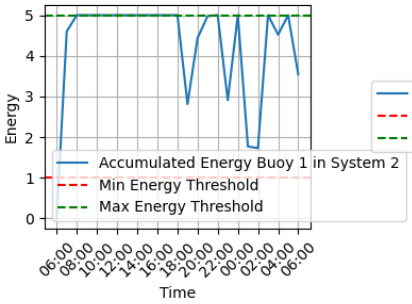

Energy
Accumulated Energy Buoy 2 in System 1
Min Energy Threshold
Max Energy Threshold
Time

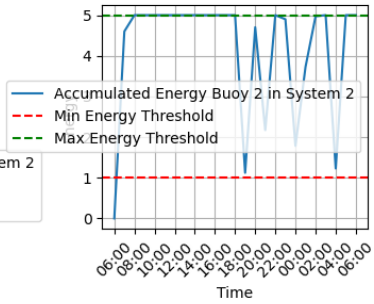

Accumulated Energy Buoy 3 in System 1
Min Energy Threshold
Max Energy Threshold
Time

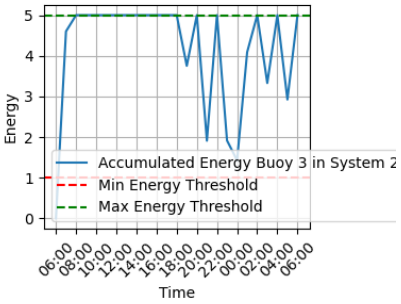

Accumulated Water Buoy 1 in System 1
Max Water Threshold
Time

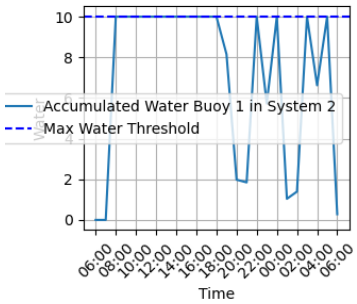

Accumulated Water Buoy 2 in System 1
Max Water Threshold
Time

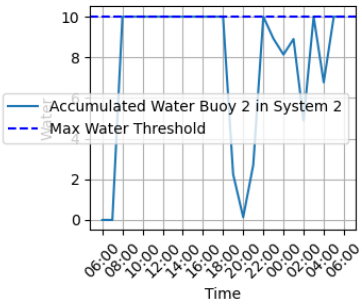

Accumulated Water Buoy 3 in System 1
Max Water Threshold
Time

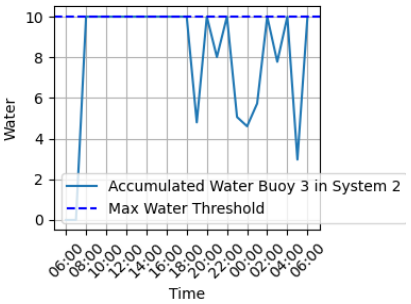

Energy
Accumulated Energy Buoy 1 in System 2
Min Energy Threshold
Max Energy Threshold
Time

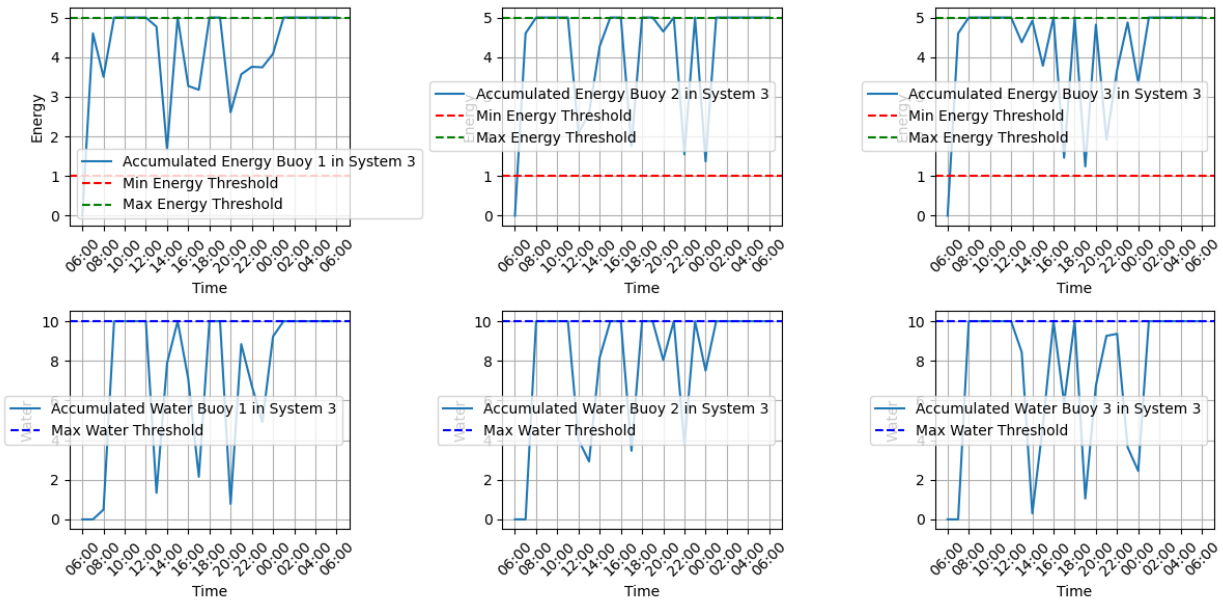

Accumulated Energy Buoy 2 in System 2
Min Energy Threshold
Max Energy Threshold
Time


Energy
Accumulated Energy Buoy 3 in System 2
Min Energy Threshold
Max Energy Threshold
Time


Accumulated Water Buoy 1 in System 2
Max Water Threshold
Time


Accumulated Water Buoy 2 in System 2
Max Water Threshold
Time


Water
Accumulated Water Buoy 3 in System 2
Max Water Threshold
Time



The provided graphs display the accumulated energy and water levels for each buoy in three autonomous buoy systems over a 24-hour simulation period. Below is an analysis of the results for each system.

System 1

Energy Levels:

- **Buoy 1**: The energy levels fluctuate significantly throughout the simulation. The energy often reaches the maximum threshold of 5 units and occasionally approaches the minimum threshold of 1 unit, indicating high energy demand and usage.
- **Buoy 2**: The energy levels start at the maximum and frequently drop close to the minimum threshold, showing similar usage patterns as Buoy 1.
- **Buoy 3**: The energy levels also exhibit significant fluctuations, reaching the maximum threshold multiple times, suggesting effective energy management and high energy consumption.

Water Levels:

- **Buoy 1**: The water levels fluctuate, often nearing the maximum threshold of 10 units but also showing significant drops, indicating frequent water usage and replenishment.
- **Buoy 2**: The water levels exhibit frequent fluctuations, similar to Buoy 1, suggesting consistent water usage by vessels.
- **Buoy 3**: The water levels show regular variations, often dropping to lower values, indicating efficient water management and high demand.

System 2

Energy Levels:

- **Buoy 1**: The energy levels start at the maximum threshold and show significant fluctuations, similar to System 1, indicating high energy demand and usage.

- **Buoy 2**: The energy levels frequently reach the maximum threshold and drop close to the minimum threshold, showing high usage.
- **Buoy 3**: The energy levels also exhibit significant fluctuations, similar to the other buoys, indicating balanced energy production and consumption.

Water Levels:

- **Buoy 1**: The water levels show considerable variation, often reaching the maximum threshold and dropping to lower values, indicating frequent usage.
- **Buoy 2**: The water levels exhibit frequent drops and replenishment cycles, similar to Buoy 1.
- **Buoy 3**: The water levels show regular variations, indicating consistent water usage and replenishment.

System 3

Energy Levels:

- **Buoy 1**: The energy levels start at a lower level and gradually reach the maximum threshold. Fluctuations are less pronounced compared to Systems 1 and 2.
- **Buoy 2**: The energy levels follow a similar pattern, gradually increasing and showing moderate fluctuations.
- **Buoy 3**: The energy levels also gradually increase and exhibit moderate fluctuations, suggesting a balanced energy usage pattern.

Water Levels:

- **Buoy 1**: The water levels quickly reach the maximum threshold and show less fluctuation, indicating a period of less water demand.
- **Buoy 2**: The water levels exhibit moderate fluctuations, indicating some periods of high water usage followed by replenishment.
- **Buoy 3**: The water levels follow a similar pattern, showing moderate fluctuations and periods of replenishment.

8.2.2 General Observations

1. **Fluctuation Patterns**: Systems 1 and 2 show significant fluctuations in energy and water levels, indicating high demand and usage by vessels. System 3 shows moderate fluctuations, suggesting a period of less demand.
2. **Threshold Compliance**: The energy and water levels generally remain within the defined thresholds, ensuring that the buoys do not exceed their storage capacities.
3. **Demand-Driven Consumption**: The frequent drops in both energy and water levels in Systems 1 and 2 suggest high and regular demand from the vessels, requiring the buoys to continuously supply resources. System 3's moderate fluctuations suggest a balanced usage pattern.

These results reflect the efficiency and responsiveness of the autonomous buoy systems in managing energy and water resources under varying demand conditions, with Systems 1 and 2 experiencing high usage and System 3 experiencing moderate usage.

8.3 Case n°3

See section 3.1 of the code at the previous link.

Global Parameters
The initial parameters are defined, including:

- The number of buoy systems (`num_autonomous_systems = 3`).
- The number of buoys per system (`n = 3`).
- The constant energy production (`cr = 4.6` units per timestep).
- The energy-water conversion parameter (`kc = 0.05`).
- The simulation duration (`timesteps = 24`).
- Initial levels of energy and water (`initial_energy = 5` and `initial_water = 10`).
- Maximum levels of energy and water (`max_energy = 5` and `max_water = 10`).
- Minimum energy threshold (`min_energy = 1`).
- Target water level (`x2_target = 10`).
- Maximum number of vessels per timestep (`max_vessels = 9`).

State Vector Initialization
The `initialize_state_vectors` function creates and initializes the state vectors for energy and water for each buoy with the updated initial conditions.

Vessel Number Generation
The `generate_num_vessels_per_timestep` function generates the number of vessels for each timestep, following a random distribution with varying rates of vessel arrivals.

Random Request Generation
The `generate_random_requests` function creates random requests for energy and water from the vessels arriving at each timestep.

Buoy State Update
The `update_buoys` function updates the energy and water levels of the buoys based on production and energy-water conversion. It ensures that the levels remain within the defined maximum thresholds.

Request Assignment
The `assign_requests_optimized` function assigns vessel requests to the buoys using an optimized greedy algorithm, which minimizes the impact on the buoy's resources.

Data Verification and Consensus
The `verify_and_consensus` function ensures that the data collected by each buoy is accurate by using a consensus mechanism, where the most common value among reported values is chosen as the verified data.

Simulation Execution
The `run_simulation_optimized` function runs the simulation for the specified number of timesteps, updating buoy states, verifying data, and assigning vessel requests at each timestep using the optimized greedy algorithm.

Results Visualization
The `plot_results_multiple_systems` function visualizes the simulation results, displaying the energy and water levels for each buoy over time.

Summary of Results

Energy Levels

- The energy levels of all buoys in each system show significant fluctuations throughout the simulation.
- The energy levels start at the maximum threshold (5 units) and fluctuate due to the varying demands from the vessels.
- The continuous rise and fall in energy levels suggest a balance between energy production and consumption by the vessels.

Water Levels

- The water levels exhibit considerable variations, starting at the maximum threshold (10 units).
- There are instances where the water levels drop to lower values, indicating high water usage by the vessels and efficient replenishment cycles.

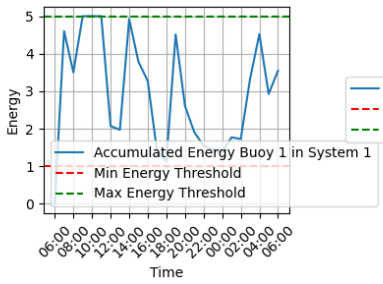
Optimized Greedy Assignment

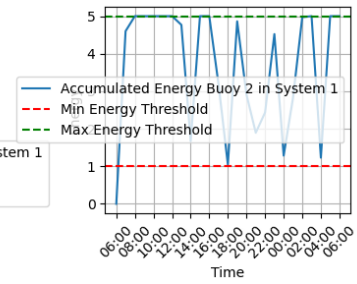
- The optimized greedy algorithm ensures efficient utilization of buoy resources by minimizing the impact of each request.
- The method effectively assigns requests while maintaining energy and water levels within defined thresholds, ensuring reliable operation.

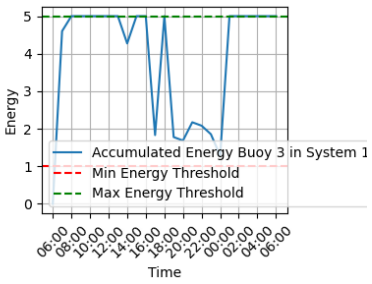
Data Verification and Consensus

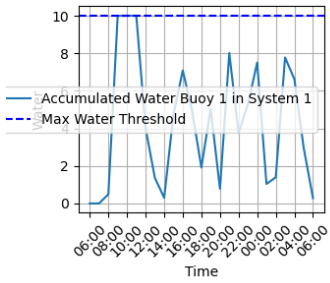
- The consensus mechanism ensures data accuracy by verifying the reported values among the buoys and updating the state vectors accordingly.
- This process helps in maintaining the integrity of the simulation data and provides robust results.

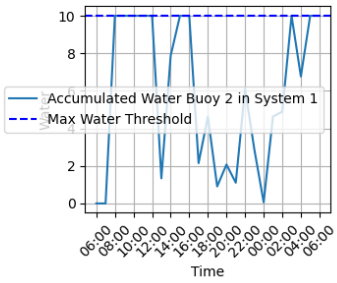
8.3.1 Results

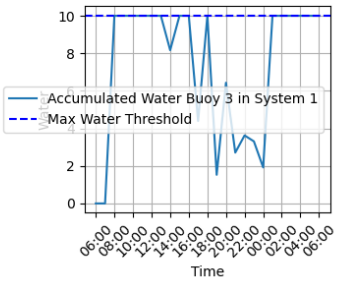

Energy
Time
Accumulated Energy Buoy 1 in System 1
Min Energy Threshold
Max Energy Threshold

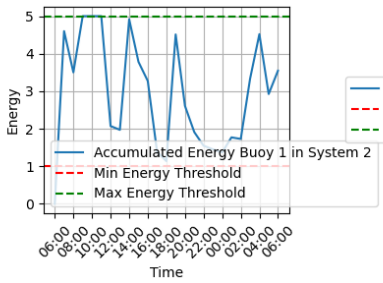

Time
Accumulated Energy Buoy 2 in System 1
Min Energy Threshold
Max Energy Threshold

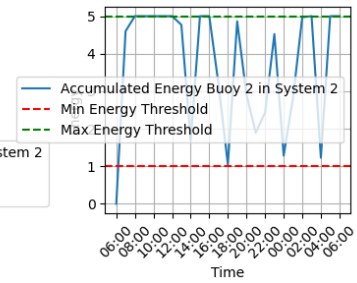

Energy
Time
Accumulated Energy Buoy 3 in System 1
Min Energy Threshold
Max Energy Threshold

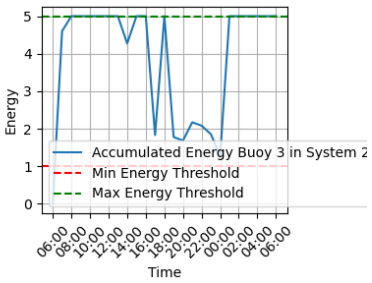

Time
Accumulated Water Buoy 1 in System 1
Max Water Threshold

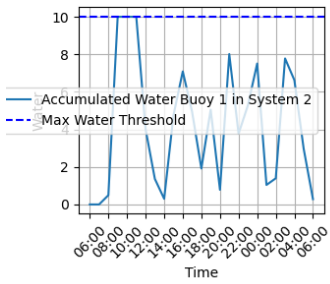

Time
Accumulated Water Buoy 2 in System 1
Max Water Threshold

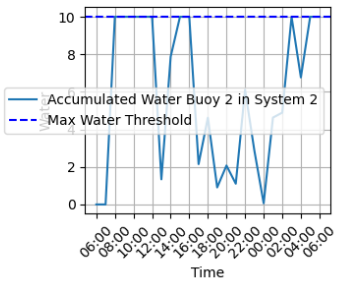

Time
Accumulated Water Buoy 3 in System 1
Max Water Threshold

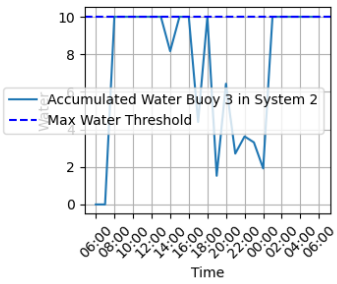

Energy
Time
Accumulated Energy Buoy 1 in System 2
Min Energy Threshold
Max Energy Threshold

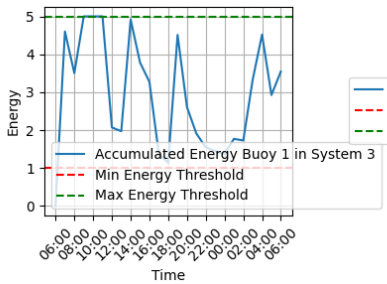

Time
Accumulated Energy Buoy 2 in System 2
Min Energy Threshold
Max Energy Threshold

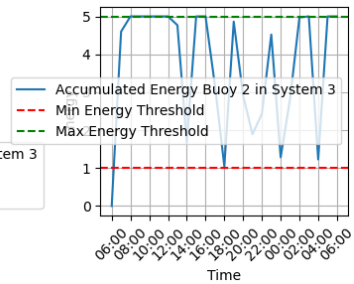

Energy
Time
Accumulated Energy Buoy 3 in System 2
Min Energy Threshold
Max Energy Threshold

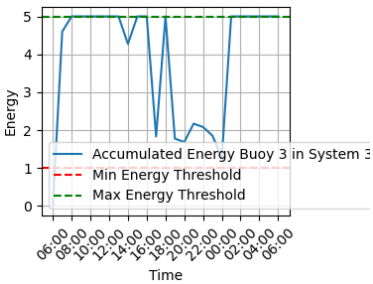

Time
Accumulated Water Buoy 1 in System 2
Max Water Threshold

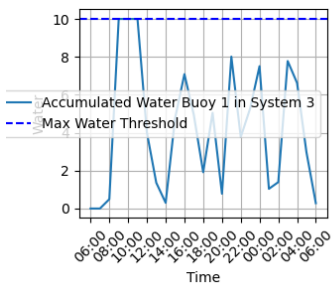

Time
Accumulated Water Buoy 2 in System 2
Max Water Threshold

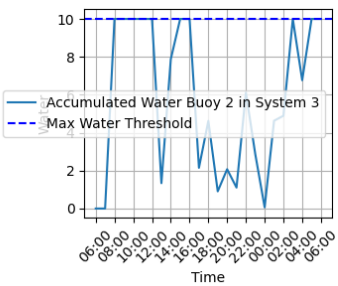

Time
Accumulated Water Buoy 3 in System 2
Max Water Threshold

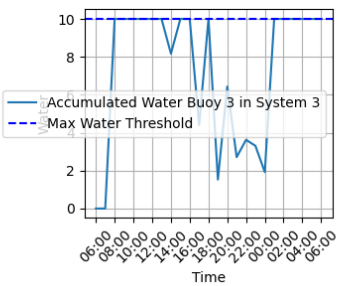

Energy
Time
Accumulated Energy Buoy 1 in System 3
Min Energy Threshold
Max Energy Threshold


Time
Accumulated Energy Buoy 2 in System 3
Min Energy Threshold
Max Energy Threshold


Energy
Time
Accumulated Energy Buoy 3 in System 3
Min Energy Threshold
Max Energy Threshold


Time
Accumulated Water Buoy 1 in System 3
Max Water Threshold


Time
Accumulated Water Buoy 2 in System 3
Max Water Threshold


Time
Accumulated Water Buoy 3 in System 3
Max Water Threshold

The provided graphs display the accumulated energy and water levels for each buoy in three autonomous buoy systems over a 24-hour simulation period. Below is an analysis of the results for each system.

System 1

Energy Levels:

- **Buoy 1**: The energy levels fluctuate significantly, often nearing the maximum threshold of 5 units and occasionally dropping close to the minimum threshold of 1 unit. This suggests high energy demand and effective usage.
- **Buoy 2**: Similar to Buoy 1, the energy levels fluctuate considerably, indicating frequent energy consumption and replenishment.
- **Buoy 3**: The energy levels show notable fluctuations, suggesting consistent energy demand and effective resource management.

Water Levels:

- **Buoy 1**: The water levels fluctuate, with the buoy often approaching the maximum water threshold of 10 units and experiencing significant drops, indicating active water usage.
- **Buoy 2**: The water levels exhibit similar patterns to Buoy 1, showing consistent fluctuations and indicating regular water demand.
- **Buoy 3**: The water levels also fluctuate, reflecting regular usage and efficient water management.

System 2

Energy Levels:

- **Buoy 1**: The energy levels start at the maximum threshold and exhibit significant fluctuations, similar to System 1, indicating high energy usage.
- **Buoy 2**: The energy levels frequently reach the maximum threshold and drop close to the minimum threshold, showing substantial usage.
- **Buoy 3**: The energy levels also show significant fluctuations, reflecting balanced energy production and consumption.

Water Levels:

- **Buoy 1**: The water levels show considerable variation, often reaching the maximum threshold and experiencing drops, indicating frequent usage.
- **Buoy 2**: The water levels exhibit frequent drops and replenishment cycles, similar to Buoy 1.
- **Buoy 3**: The water levels show regular variations, indicating consistent water usage and replenishment.

System 3

Energy Levels:

- **Buoy 1**: The energy levels start at a lower level and quickly reach the maximum threshold. Fluctuations are less pronounced compared to Systems 1 and 2, suggesting lower demand.
- **Buoy 2**: The energy levels follow a similar pattern, quickly reaching and showing moderate fluctuations.

- **Buoy 3**: The energy levels also quickly increase and exhibit moderate fluctuations, indicating balanced energy usage.

Water Levels:

- **Buoy 1**: The water levels quickly reach the maximum threshold and show less fluctuation, indicating a period of less water demand.
- **Buoy 2**: The water levels exhibit moderate fluctuations, indicating periods of high water usage followed by replenishment.
- **Buoy 3**: The water levels follow a similar pattern, showing moderate fluctuations and periods of replenishment.

8.3.2 General Observations

1. **Fluctuation Patterns**: Systems 1 and 2 show significant fluctuations in energy and water levels, indicating high demand and usage by vessels. System 3 shows moderate fluctuations, suggesting a period of less demand.
2. **Threshold Compliance**: The energy and water levels generally remain within the defined thresholds, ensuring that the buoys do not exceed their storage capacities.
3. **Demand-Driven Consumption**: The frequent drops in both energy and water levels in Systems 1 and 2 suggest high and regular demand from the vessels, requiring the buoys to continuously supply resources. System 3's moderate fluctuations suggest a balanced usage pattern.

These results reflect the efficiency and responsiveness of the autonomous buoy systems in managing energy and water resources under varying demand conditions, with Systems 1 and 2 experiencing high usage and System 3 experiencing moderate usage.

8.4 Case n°4

See section 3.2 of the code at the previous link.

Global Parameters
The initial parameters are defined, including:

- The number of buoy systems (`num_autonomous_systems = 3`).
- The number of buoys per system (`n = 3`).
- The constant energy production (`cr = 4.6` units per timestep).
- The energy-water conversion parameter (`kc = 0.05`).
- The simulation duration (`timesteps = 24`).
- Initial levels of energy and water (`initial_energy = 1` and `initial_water = 0`).
- Maximum levels of energy and water (`max_energy = 5` and `max_water = 10`).
- Minimum energy threshold (`min_energy = 1`).
- Target water level (`x2_target = 10`).
- Maximum number of vessels per timestep (`max_vessels = 9`).

State Vector Initialization
The `initialize_state_vectors` function creates and initializes the state vectors for energy and water for each buoy with the specified initial conditions.

Vessel Number Generation
The `generate_num_vessels_per_timestep` function generates the number of vessels for each timestep, following a random distribution with varying rates of vessel arrivals.

Random Request Generation
The `generate_random_requests` function creates random requests for energy and water from the vessels arriving at each timestep.

Buoy State Update
The `update_buoys` function updates the energy and water levels of the buoys based on production and energy-water conversion. It ensures that the levels remain within the defined maximum thresholds.

Request Assignment
The `assign_requests_to_systems` function assigns vessel requests to the buoys within each system. It uses the `assign_requests_to_buoys` function to optimize the assignment by minimizing the impact on buoy resources.

Data Verification and Consensus
The `verify_and_consensus` function ensures that the data collected by each buoy is accurate by using a consensus mechanism, where the most common value among reported values is chosen as the verified data.

Simulation Execution
The `run_simulation_optimized` function runs the simulation for the specified number of timesteps, updating buoy states, verifying data, and assigning vessel requests at each timestep using the optimized assignment algorithm.

Results Visualization
The `plot_results_multiple_systems` function visualizes the simulation results, displaying the energy and water levels for each buoy over time. It includes labels for each timestep, formatted as hours of the day.

Summary of Results
Energy Levels

- The energy levels of all buoys in each system show significant fluctuations throughout the simulation.
- The energy levels start at a lower initial level (1 unit) and fluctuate due to the varying demands from the vessels.
- The continuous rise and fall in energy levels suggest a balance between energy production and consumption by the vessels.

Water Levels

- The water levels exhibit considerable variations, starting at the initial level (0 units).
- There are instances where the water levels increase due to the conversion of energy to water, indicating efficient replenishment cycles.

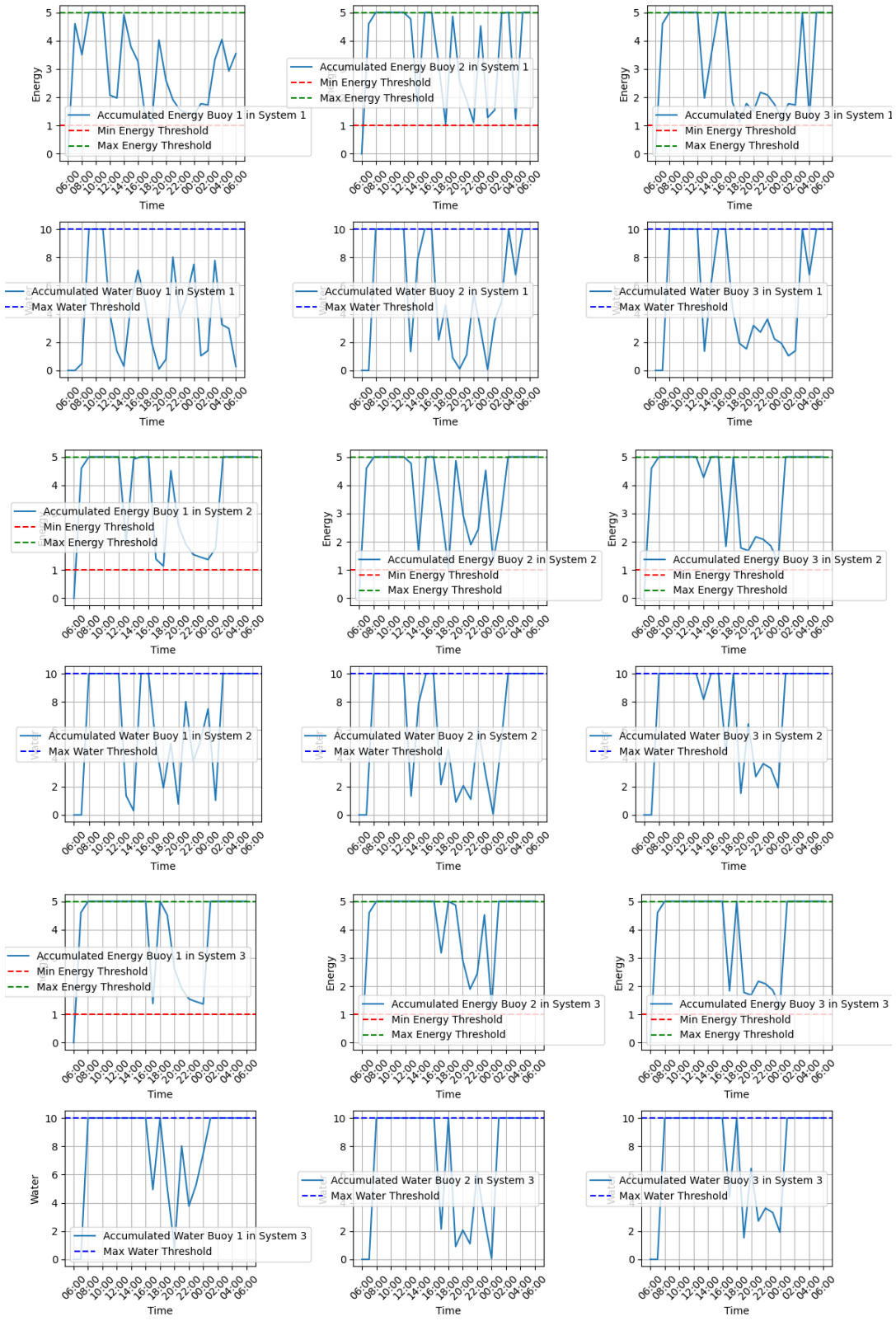
Optimized Assignment

- The optimized assignment algorithm ensures efficient utilization of buoy resources by minimizing the impact of each request.
- The method effectively assigns requests while maintaining energy and water levels within defined thresholds, ensuring reliable operation.

Data Verification and Consensus

- The consensus mechanism ensures data accuracy by verifying the reported values among the buoys and updating the state vectors accordingly.
- This process helps in maintaining the integrity of the simulation data and provides robust results.

8.4.1 Results



The provided graphs display the accumulated energy and water levels for each buoy in three autonomous buoy systems over a 24-hour simulation period. Below is an analysis of the results for each system.

System 1

Energy Levels:

- **Buoy 1:** The energy levels fluctuate significantly, often nearing the maximum threshold of 5 units and occasionally dropping close to the minimum threshold of 1 unit. This suggests high energy demand and effective usage.
- **Buoy 2:** Similar to Buoy 1, the energy levels fluctuate considerably, indicating frequent energy consumption and replenishment.
- **Buoy 3:** The energy levels show notable fluctuations, suggesting consistent energy demand and effective resource management.

Water Levels:

- **Buoy 1:** The water levels fluctuate, with the buoy often approaching the maximum water threshold of 10 units and experiencing significant drops, indicating active water usage.
- **Buoy 2:** The water levels exhibit similar patterns to Buoy 1, showing consistent fluctuations and indicating regular water demand.
- **Buoy 3:** The water levels also fluctuate, reflecting regular usage and efficient water management.

System 2

Energy Levels:

- **Buoy 1:** The energy levels start at the maximum threshold and exhibit significant fluctuations, similar to System 1, indicating high energy usage.
- **Buoy 2:** The energy levels frequently reach the maximum threshold and drop close to the minimum threshold, showing substantial usage.
- **Buoy 3:** The energy levels also show significant fluctuations, reflecting balanced energy production and consumption.

Water Levels:

- **Buoy 1:** The water levels show considerable variation, often reaching the maximum threshold and experiencing drops, indicating frequent usage.
- **Buoy 2:** The water levels exhibit frequent drops and replenishment cycles, similar to Buoy 1.
- **Buoy 3:** The water levels show regular variations, indicating consistent water usage and replenishment.

System 3

Energy Levels:

- **Buoy 1:** The energy levels start at a lower level and quickly reach the maximum threshold. Fluctuations are less pronounced compared to Systems 1 and 2, suggesting lower demand.
- **Buoy 2:** The energy levels follow a similar pattern, quickly reaching and showing moderate fluctuations.
- **Buoy 3:** The energy levels also quickly increase and exhibit moderate fluctuations, indicating balanced energy usage.

Water Levels:

- **Buoy 1:** The water levels quickly reach the maximum threshold and show less fluctuation, indicating a period of less water demand.
- **Buoy 2:** The water levels exhibit moderate fluctuations, indicating periods of high water usage followed by replenishment.
- **Buoy 3:** The water levels follow a similar pattern, showing moderate fluctuations and periods of replenishment.

8.4.2 General Observations

- **Fluctuation Patterns:** Systems 1 and 2 show significant fluctuations in energy and water levels, indicating high demand and usage by vessels. System 3 shows moderate fluctuations, suggesting a period of less demand.
- **Threshold Compliance:** The energy and water levels generally remain within the defined thresholds, ensuring that the buoys do not exceed their storage capacities.
- **Demand-Driven Consumption:** The frequent drops in both energy and water levels in Systems 1 and 2 suggest high and regular demand from the vessels, requiring the buoys to continuously supply resources. System 3's moderate fluctuations suggest a balanced usage pattern.

These results reflect the efficiency and responsiveness of the autonomous buoy systems in managing energy and water resources under varying demand conditions, with Systems 1 and 2 experiencing high usage and System 3 experiencing moderate usage.

9 Observations and Conclusions

In this report, we analyzed the issue of buoy assignment by considering various types of request allocations from different boats in multiple scenarios. We aimed to describe this issue as realistically as possible by modeling the random arrival of boats and distributing requests in a weighted manner over a 24-hour period.

We observed the convergence of resources (energy and water) according to different cycles of consumption and recharge at hourly intervals (each time step k equal to one hour). Additionally, we enhanced our code to maximize resource consumption on each buoy and increased the robustness of information exchange using the Byzantine Generals problem.

For future improvements, it would be beneficial to provide high availability in data exchange between autonomous systems (AS), potentially by increasing the number of generals within each AS to ensure greater robustness. Considering the geolocation of each boat is crucial; it would be impractical for a boat to be redirected to a distant buoy. The redirection should thus consider both the maximization of resource consumption per buoy and the proximity between buoy and boat.

Other important considerations include the information exchange protocols between buoys, boats, and AS, as well as the management of exceptional circumstances related to the technology used by the involved entities. The advantage of having AS with a smaller number of buoys lies in the faster convergence in decision-making compared to a fully distributed system (complete graph) with the constraint of distances between AS.

Each AS should be viewed as a unique context that communicates with other AS to provide the best possible service to the boat, as requests may randomly reach different AS. By adjusting the parameters of the models developed in the simulations, we can stress test the system in an ideal environment to understand the maximum

load between resources and requests. This will ensure the evolution of the model and its development in a real-world context.