Fundamentals of Adaptive Software
Assignment 3

Fundamentals of Adaptive Software 2023

Group: 5.1
Member names: Cristian - Augustin Susanu, Sree Padmavathy Balaji, Yuktha Sri, Dalvie Benu
Emails: c.susanu@student.vu.nl, s.p.balaji@student.vu.nl, y.s.chittoor.mukkoteeswaran@student.vu.nl, d.c.benu@student.vu.nl
VUnetIDs: csu100, sba467, ych104, dbe297
Master program and Track: Computer Science - SEG

March 9, 2024

Contents

1 Introduction

1.1 About the Exemplar

"DingNet: A Self-Adaptive Internet-of-Things Exemplar" offers an insightful exploration into the world of self-adaptive systems within the Internet of Things (IoT), particularly in smart city applications. The work introduces DingNet, an advanced simulator replicating the behavior of IoT networks in urban settings, focusing on the interactions of mobile and stationary devices connected through a LoRaWAN network.

The initial sections outline DingNet's significance in IoT, emphasizing the need for self-adaptive mechanisms to handle complexities and uncertainties in smart cities. This sets the foundation for understanding the challenges and potential solutions in IoT environments.

The core of the study is the technical details of the DingNet simulator. It describes the system's architecture, illustrating how it simulates real-world IoT scenarios. The exemplar simulates the mobile end nodes (motes), gateways as well as the application servers. DingNet also simulates the LoRaWAN communication between the motes and gateways and the environmental noise that affect the communications. The exemplar simulates the running of the IoT in a defined environment as the motes moves from one point to another then ends.As the motes travel, it sends information to the gateways that may be affected by environmental noise.

A case study based on Leuven, Belgium, demonstrates DingNet's practical application, highlighting its utility in modeling and optimizing IoT networks in urban areas. This practical example offers insights into the challenges and advantages of implementing DingNet in smart city contexts.

In conclusion, the work reflects on DingNet's contributions to IoT and self-adaptive systems, underscoring its role in advancing the understanding of these technologies. The study closes by suggesting future research directions, expanding DingNet's application beyond smart cities. [1]

1.2 Overall Description of Adaptive System

The section provides a comprehensive overview of an adaptive system designed for DingNet IoT environments, particularly in smart cities. It addresses the unpredictability of environmental conditions affecting network reliability and the system requirements that ensure communication quality and responsiveness. The PID controller is introduced as the preferred solution for real-time adaptation, chosen for its precision and simplicity over the computationally intensive Q-learning. The strategy behind the system's design, implementation, and the advantages of using a PID controller are also discussed, demonstrating the system's capability to maintain robust communication in the face of environmental uncertainties.

- **Uncertainty Analysis:** The Dingnet's system's uncertainty analysis delves into the unpredictable nature of environmental conditions that can severely affect IoT communication. It identifies how variables like weather patterns, urban development, and topological changes can cause signal degradation, leading to increased packet loss and energy use. The analysis underlines the importance of an adaptable network that can respond to these changes without compromising on the critical performance metrics of reliability and energy efficiency.

- **System Requirements:** The system requirements section outlines the objectives for the IoT framework in Dingnet, emphasizing the need for maintaining consistent communication quality amid environmental uncertainties. It mandates the ability of the system to dynamically adjust transmission power and spreading factor of network nodes to address signal degradation. Crucially, the requirements specify that packet loss must be kept below 5 percent to ensure operational continuity, prompt environmental responsiveness, and the protection of data integrity, thus preventing any service disruptions.

- **Potential Solutions:** In the search for an effective adaptation mechanism, the Proportional-Integral-Derivative (PID) controller is pinpointed as the ideal solution. Its real-time response capabilities make it a practical tool for adjusting transmission power and spreading factor in response to environmental factors, a critical requirement for maintaining network stability. The PID controller's strengths in precision control, rapid response, and reliability make it a preferred choice over Q-learning, which, despite its potential for optimized decision-making, is sidelined due to its complexity and heavy demands on system resources.

- **Adaptation Strategy Design:** The adaptation strategy harnesses the PID controller's sophisticated mechanism of dynamically modulating power and spreading factor levels based on environmental feedback, ensuring optimal communication quality. This design is meticulously structured to provide a reliable network operation that adapts swiftly to changing conditions. It is a strategic move towards creating a resilient IoT ecosystem that not only meets the immediate demands of network reliability but also contributes to the long-term sustainability of smart city infrastructure.

- **Implementation and Showcase:** The implementation of the PID controller within the IoT infrastructure is demonstrated through practical applications. Its real-time tuning of transmission power and spreading factor in response to varying environmental signals showcases an efficient and adaptable network system. This practical demonstration provides tangible evidence of the system's ability to maintain high-quality communication and manage energy usage effectively.

- **Evaluation:** The evaluation contrasts the adaptive PID system with static, non-adaptive strategies, highlighting its robustness. The superiority of the PID approach is evident in its consistent performance despite environmental challenges, confirming the system's resilience and validating the design choices made during the system's conceptualization.

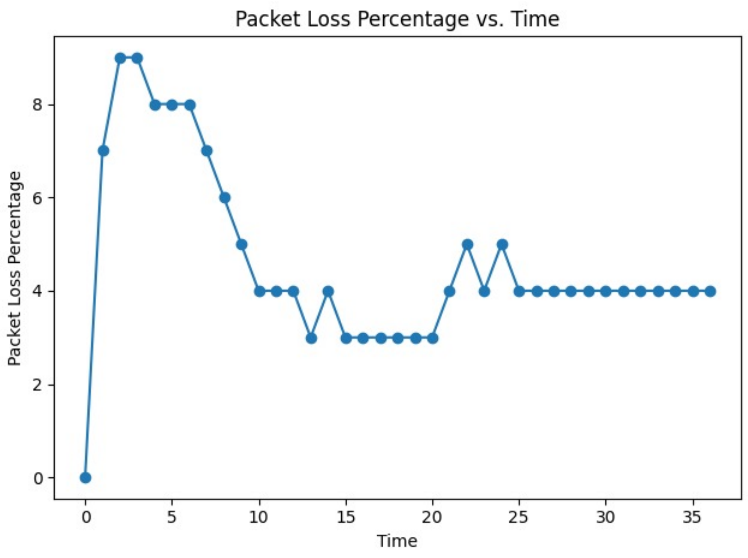


Figure 1: Packet Loss(in percentage) Vs Time

The graph indicates an initial high packet loss percentage that sharply decreases over time and then stabilizes. In our adaptive strategy, this trend could suggest that the system has successfully implemented dynamic adjustments to mitigate packet loss. The initial spike may represent the onset of environmental stressors, with the subsequent decline indicating DingNet's real-time response in adjusting network parameters to enhance data transmission reliability. The stabilization at a lower packet loss percentage reflects the effectiveness of our adaptation strategy over time.

2 Analysis of uncertainties

The integration of adaptive systems within Internet-of-Things (IoT) architectures necessitates a comprehensive understanding of the uncertainties that can impact system performance. In the context of this report, we examine the dynamic environmental factors that pose a significant challenge to network reliability. The table presented synthesizes the various dimensions of this uncertainty, ranging from its definition and classification to the strategic responses employed to mitigate its adverse effects. Through an adaptive management approach, we aim to maintain optimal service quality and efficient energy usage despite environmental fluctuations, ensuring robust communication between network motes and gateways.

Field	**Description**
Name	Environmental Dynamics Impacting Network Reliability
Classification	This uncertainty is classified as an external operational factor occurring during execution that influences system performance without direct control from the system's internal processes.

Context	The source of uncertainty is changing weather conditions and the conditions under which it occurs include a range of environmental factors such as variable weather patterns, urban development, and changing topologies that can disrupt signal propagation and energy management.
Impact	Fluctuating environmental conditions can lead to increased packet loss as signals degrade or are obstructed, necessitating additional energy expenditure to boost signal strength or retransmit data, ultimately impacting the system's energy efficiency and communication reliability.
Degree of severity	Ranges from minor, causing negligible disruptions, to severe, potentially leading to system outages, increased packet loss or significant energy drains.
Adaptation goals	The primary objective is to uphold an optimal level of quality of service and limit the packet loss, while conserving energy despite the fluctuations in environmental conditions.
Mitigation Strategies	The mitigation strategy focuses on the deployment of adaptive algorithms capable of modulating transmission power and spreading factor levels in direct response to the fluctuations in signal quality. This approach entails a dynamic assessment of real-time communication conditions, allowing the system to conserve energy by reducing transmission power and spreading factor when signal quality is high, and conversely, increasing them to ensure data integrity when faced with poor signal conditions. The goal is to achieve a balance between reliable communication and energy efficiency, adapting the power and spreading factor used for transmission as environmental conditions dictate.
Sample Illustration	A sensor network deployed in an urban environment may encounter increased packet loss during a heavy rainstorm due to signal attenuation. The network's adaptive protocol can counteract this by increasing the transmission power and spreading factor, ensuring data integrity without undue energy waste.
Evaluation	An evaluative comparison of system resilience with adaptive measures in place versus a non-adaptive configuration under simulated environmental stressors.
Also Known As	This uncertainty is also referred to as "Environmental Variability" or "Operational Context Changes," highlighting its nature as an external influence that requires the system's operational flexibility.

Table 1: Environmental Uncertainty

2.1 Need for Addressing the Uncertainty

In the context of DingNet, a simulator for IoT systems, addressing environmental factors is crucial for ensuring the system's functionality. Specifically, in urban traffic control applications, the reliability of sensor data transmission is vital. Signal attenuation caused by adverse weather conditions, such as dense fog or heavy rain, could lead to data loss, negatively impacting traffic management's real-time operations, city traffic flow, and public safety. DingNet's adaptive capabilities are therefore essential for preserving communication effectiveness in such challenging scenarios.

For example, if a traffic sensor fails to send critical congestion data due to poor signal quality because of environmental variability, the result could be sub optimal traffic light control leading to increased congestion and even accidents. Hence, understanding and mitigating such environmental uncertainties is essential to ensure that the data remains reliable and that systems can react appropriately to real-time conditions.

2.2 Importance of Addressing the Uncertainty

In the context of DingNet and smart city applications, addressing environmental uncertainties is crucial. For example, in urban traffic management, sensors within DingNet's network must reliably transmit congestion levels and traffic patterns. Signal loss due to environmental factors like dense fog could disrupt traffic flow management. Additionally, sensor-driven waste management systems require accurate data to optimize collection routes and schedules.

Addressing uncertainties in smart city applications is vital for ensuring the robustness and reliability of IoT systems like DingNet. The ability to adapt to environmental factors, such as variable weather conditions, is essential for maintaining the integrity of data-driven services. Effective management of these uncertainties safeguards against disruptions in critical urban infrastructure, enhancing the quality of life and operational efficiency within smart cities.Adaptive strategies within DingNet are essential to prevent such data loss, ensuring efficient city operations and contributing to the overall sustainability of urban living.

3 Requirements

This encompasses both traditional requirements, which outline the expected steadfast functionalities and performance thresholds under any operational conditions, and RELAX requirements, which introduce flexibility to adapt to environmental uncertainties. Together, these requirements frame the system's expected performance, including its responsiveness to fluctuating transmission power needs to maintain communication reliability and manage energy consumption effectively.

3.1 Traditional Requirements [2]

- The system **SHALL** maintain a constant transmission power and spreading factor levels to ensure communication reliability under all environmental conditions.
- The communication system **SHALL NOT** experience a packet loss rate exceeding 0.05 percentage regardless of signal strength fluctuations due to environmental factors.
- The system **SHALL** guarantee data packet delivery within a predefined time frame, irrespective of varying weather patterns or urban development changes.
- The energy consumption of the network **SHALL** remain within the specified operational budget for power usage, without adjustments for environmental context.
- The system **SHALL** utilize a uniform algorithm for signal transmission that does not alter based on environmental stimuli or detected signal interference.
- The network **SHALL** have a fail-safe operational mode that activates during adverse environmental conditions, ensuring no degradation in communication quality.

3.2 RELAX Requirements [2]

- The system **SHOULD** generally maintain a stable transmission power level and spreading factor, but it **MAY ADJUST** them in response to significant changes in environmental conditions to preserve communication reliability.
- The communication system **SHOULD** strive to minimize packet loss to 5 percent, but the packet loss rate **CAN INCREASE** during extreme environmental interference, provided it returns to acceptable levels once conditions improve.
- Data packet delivery **SHOULD** be timely under normal conditions; however, the system **CAN ADAPT** delivery times when environmental factors such as weather fluctuations or urban development interfere with transmission paths.
- While energy consumption **SHOULD** remain within target ranges, the system **IS PERMITTED TO EXCEED** these limits temporarily to cope with adverse environmental changes, with the expectation of returning to normal levels post-adaptation.
- The network **SHOULD NORMALLY** avoid using high transmission power and spreading factor to conserve energy, but it **MAY INCREASE** them when signal degradation is detected.
- Sensor data **SHOULD BE TRANSMITTED** reliably at regular intervals, but intervals **CAN BE ADJUSTED** if environmental conditions threaten data integrity.

4 Analysis of potential solutions

This section provides a critical comparison between two adaptation strategies for Dingnet and smart city applications: a Proportional-Integral-Derivative (PID) controller and an Adaptive Power Modulation Strategy using Q-learning. The PID controller is favored for its precise control, adaptability, and simplicity, which are key in managing the dynamic environmental variations affecting signal integrity. Conversely, Q-learning, while offering a theoretically optimized decision-making process, is rejected due to its complexity, long convergence time, and high computational demand, which are impractical for real-time system constraints.

4.1 Solution 1

Utilizing a Proportional-Integral-Derivative (PID) controller

In the quest to enhance system reliability and efficiency in dynamic IoT and smart city environments, we have adopted a Proportional-Integral-Derivative (PID) controller as our preferred solution. This approach is anchored in the controller's capacity for precision adjustments to transmission power and spreading factor, critical factors in mitigating signal decay and minimizing packet loss. The PID controller's robust and adaptable nature, combined with its straightforward design, offers a dependable means to ensure continuous and stable operation, demonstrating a significant improvement over previous methods which could not consistently address the nuances of environmental variability.

Status : Accepted

4.1.1 Evaluation of the solution with ATAM technique

ATAM : Architecture Tradeoff Analysis Method

Scenario Name	Dynamic Transmission Adaptation
Attributes(s)	Control Precision, Stability, Adaptability, Speed of Response, Robustness, Simplicity, Communication Integrity.
Environment	IoT and smart city applications where dingnet is deployed with fluctuating environmental conditions.
Stimulus	Environmental variations causing signal decay.
Response	PID controller adjusts transmission power dynamically to reduce packet loss.
Architectural decisions	• PID Controller for transmission power and spreading factor control. • Tuning of Proportional, Integral, Derivative parameters. • Adaptation logic to environmental feedback.
Sensitivity	• Transmission power and spreading factor adaptation rate. • Spreading factor adaptation rate. • Environmental change detection sensitivity.

TradeOff	• Transmission power and spreading factor control precision vs. energy consumption. • Spreading factor control precision vs. energy consumption. • Adaptation speed vs. signal overshooting risk.
Risk	• Slow adaptation causing intermittent connectivity.
NonRisk	• The likelihood of the system not responding to control adjustments is low, given the PID controller's proven track record in various applications. • The risk of the controller not compensating for steady-state errors is minimal due to the integral action within the PID. • The risk of delayed system response is mitigated by the derivative component of the PID, which anticipates system behavior.
Risk Themes	• Optimizing energy efficiency while ensuring reliable communication. • Achieving a balance between swift adaptation and long-term system endurance.
Reasoning	• PID control's reliable response to fluctuating signal conditions helps maintain communication fidelity. • The balance between fast adaptation and energy conservation is critical for sustainable operation. • Recognizing non-risks assures stakeholders of the system's reliability and hardware longevity despite adaptive power adjustments.

Table 2: ATAM evaluation table - Solution 1

4.1.2 Rationale for accepting the solution

- **Precision Control:** PID controllers excel in fine-tuning system output, adeptly modifying transmission power and spreading factor to address signal decay and minimize packet loss, ensuring data integrity even with environmental interferences.
- **System Stability:** They are crucial for maintaining operational consistency amidst the dynamic and unpredictable variables present in smart city ecosystems, effectively compensating for environmental fluctuations.
- **Adaptive Modulation:** The integral function of the PID controller integrates errors over time, adeptly adjusting to sustained signal variations and preserving communication flow.

- **Prompt Response:** With the derivative feature, PID controllers forecast potential signal disruptions, allowing for preemptive adjustments to the transmission power and spreading factor, vital for real-time communication systems.
- **Robust Performance:** Renowned for their durability, PID controllers reliably function across a spectrum of conditions, ensuring minimal disruption to IoT networks due to environmental changes.
- **Simplicity and Dependability:** Their straightforward architecture not only simplifies implementation but also guarantees consistent performance, as evidenced by scenario analyses demonstrating reduced packet loss in varying signal conditions.

4.1.3 Advantages of the chosen PID controller solution

- Reliable Communication: Ensures consistent data transmission by dynamically adjusting power and spreading factor to counteract signal decay, vital in maintaining IoT connectivity.
- Energy Efficiency: The PID controller optimizes the tradeoff between maintaining signal strength and conserving energy, crucial for sustainable smart city operations.

4.1.4 Disadvantages of the chosen PID controller solution

- Complex Tuning: Requires precise calibration of parameters, which can be complex and time-consuming to achieve optimal performance.
- Potential Overshooting: There's a risk of overshooting the desired signal strength, leading to possible energy waste and interference with other systems.

4.2 Solution 2

Adaptive Power Modulation Strategy using Q-learning

The Adaptive Power Modulation Strategy using Q-learning was explored as a sophisticated approach to enhance our system's adaptability and decision-making processes through environmental interaction. However, despite its potential for refined strategy development based on historical data, this solution was ultimately dismissed. The complexity and demand for extensive data, coupled with the algorithm's extensive convergence time and computational intensity, rendered it unsuitable for our real-time operational needs, where promptness and system resource conservation are critical.

Status : Rejected

4.2.1 Evaluation of the solution with ATAM technique

ATAM : Architecture Tradeoff Analysis Method

Scenario Name	Adaptive Power Modulation Strategy using Q-learning
Attributes(s)	Control Precision, Stability, Adaptability, Speed of Response, Robustness, Simplicity, Communication Integrity.

Environment	IoT and smart city applications where dingnet is deployed with fluctuating environmental conditions.
Stimulus	Environmental variations causing signal decay.
Response	A system that learns and adapts from environmental interactions in real-time using q learning.
Architectural decisions	• Utilization of Q-learning for adaptive learning. • Data analysis for environmental interaction patterns. • Real-time system feedback for continuous learning.
Sensitivity	• Algorithm's learning rate. • Data input frequency and quality. • Reward function calibration.
TradeOff	• Learning efficiency vs. computational overhead. • Long-term adaptability vs. short-term performance stability.
Risk	• Overfitting to noisy data. • Underfitting due to insufficient training. • Delays in learning affecting real-time responsiveness.
NonRisk	• System instability due to adaptive learning • Excessive energy consumption for learning processes
Risk Themes	• Balancing computational efficiency with learning effectiveness. • Ensuring timely adaptation to maintain system performance.
Reasoning	Adopting Q-learning involves carefully balancing the speed and accuracy of learning against the resources it consumes, ensuring the system adapts effectively without compromising other operations.

Table 3: ATAM evaluation table - Solution 2

4.2.2 Rationale for rejecting the solution

We considered Q-learning for our adaptation strategy due to its potential to optimize decision-making by learning from interactions with the environment. However, we rejected Q-learning for the following reasons:

- **Complexity:** Q-learning requires a large amount of data and numerous iterations to learn effectively, which may not be practical for our system with real-time constraints.
- **Convergence Time:** It can take a significant amount of time for the algorithm to converge to an optimal policy, especially in a complex and high-dimensional space.
- **Exploration vs. Exploitation**: Balancing between exploring new strategies and exploiting known good strategies is challenging and can lead to suboptimal performance during the learning phase.
- **Resource Intensity:** Q-learning is computationally intensive, which could strain the system's resources, potentially affecting other critical operations.
- **Predictability and Safety:** Q-learning's exploratory actions might be unpredictable, which can be a concern in environments where safety and reliability are paramount.

4.2.3 Advantages of Q-learning Solution:

- Potential for optimized decision-making through environmental learning.
- Ability to adapt strategies based on historical interaction data.

4.2.4 Disadvantages of Q-learning Solution:

- Requires extensive data and numerous iterations, impractical for real-time systems.
- Long convergence time, not suitable for environments needing quick adaptation.
- Computational intensity could impact system resources and other operations.

5 Design of the proposed adaptation strategy

This section aims to further present the adaption strategy and the logic behind it.

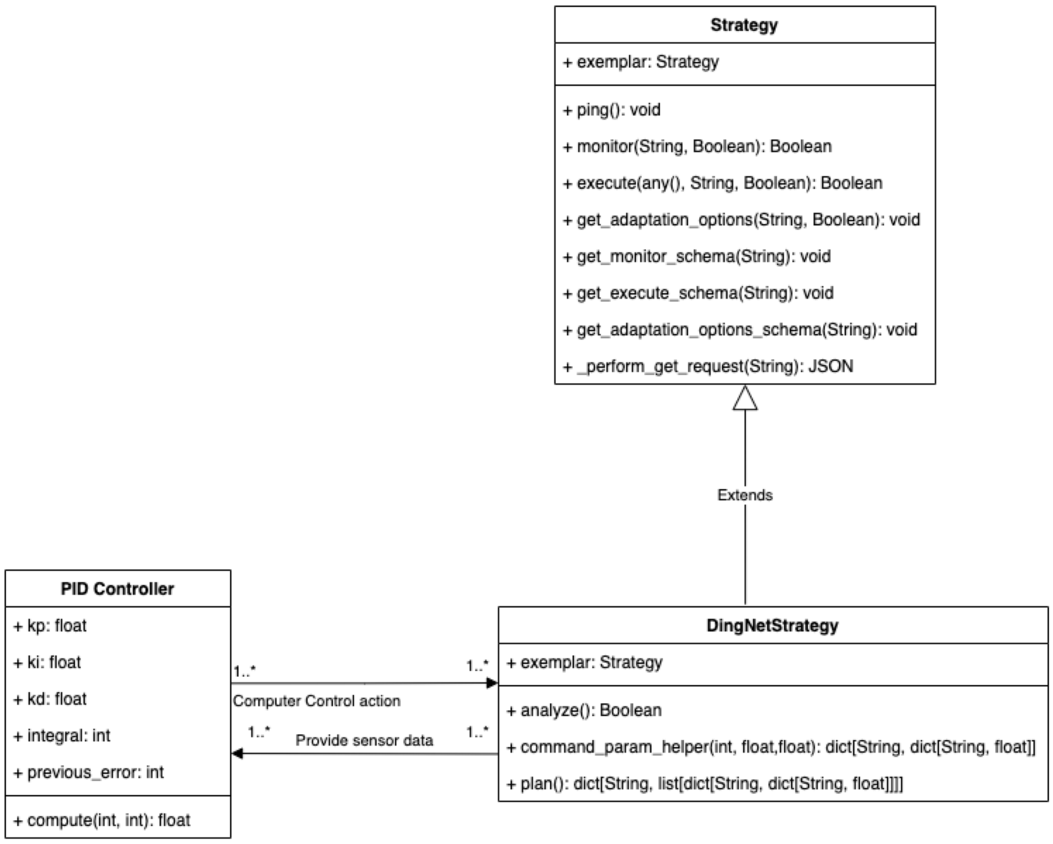

Strategy
+ exemplar: Strategy
+ ping(): void
+ monitor(String, Boolean): Boolean
+ execute(any(), String, Boolean): Boolean
+ get_adaptation_options(String, Boolean): void
+ get_monitor_schema(String): void
+ get_execute_schema(String): void
+ get_adaptation_options_schema(String): void
+ _perform_get_request(String): JSON
Extends
PID Controller
+ kp: float
+ ki: float
+ kd: float
+ integral: int
+ previous_error: int
+ compute(int, int): float
1..*
1..*
Computer Control action
1..*
Provide sensor data
1..*
DingNetStrategy
+ exemplar: Strategy
+ analyze(): Boolean
+ command_param_helper(int, float,float): dict[String, dict[String, float]]
+ plan(): dict[String, list[dict[String, dict[String, float]]]]

Figure 2: Upisas Adaption Strategy Class diagram

Picture 2 represents a class diagram depicting the main driver classes for the strategy adaption implemented in Upisas. The class that constitues the main driver is DingNetStrategy, which uses all the other classes and their respective functions to implement the adaption. This class extends on the Strategy class, which takes an exemplar as a parameter for initialization, being DingNet for the purpose of this paper. In terms of methods the Strategy class contains the main HttpServer endpoints (monitor(String, Boolean), execute(any(), String, Boolean), get_adaption_options(String, Boolean), get_monitor_schema(String), get_execute_schema(String), get_adaption_options_schema(String)). The ping() method performs a ping operation for the base endpoint of the HttpServer (found at the path: "/", representing the returned logic of the _perform_get_request() method) meaning to calculate the time it takes for a small data set to be transmitted from the device to the server on the Internet and back to your device again.

In terms of the adaption logic implemented, the DingNetStrategy has the following methods:

- analyze() - Fig.7: which checks whether packet loss is above 5%, case in which it applies the compute() method prom the PID Controller class (and subsequently Eq. 1) for each mote's power and spreading factor variables. In case the packet loss is smaller than 5%, it implies the system is working accordingly and it tries to optimize power usage by incrementally decreasing each mote's power level by 0.2 and spreading factor by 0.5.

- command_param_helper() - Fig.6: prepares the command_param_helper dictionary which constitues the base for building the JSON file to be used as a request body for the execute() function, used for updating sensor information.
- plan() - Fig.8: for each mote it updates the power level and spreading factor variables through a JSON object.

The PID controller class aims takes the parameters respective to the Proportional Integral Derivative equation presented in Eq.1 and implements it.

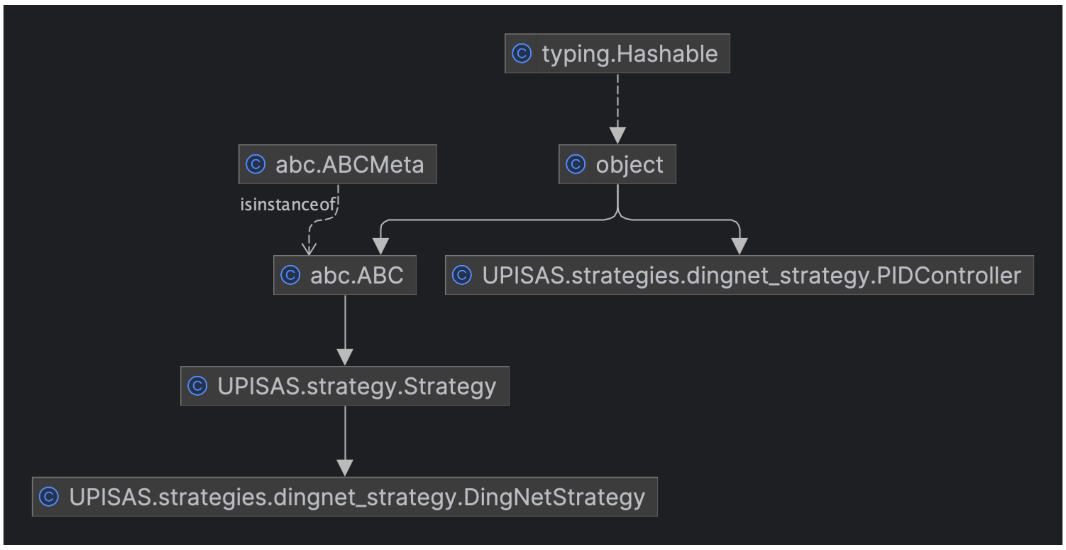

typing.Hashable
abc.ABCMeta
object
isinstanceof
abc.ABC
UPISAS.strategies.dingnet_strategy.PIDController
UPISAS.strategy.Strategy
UPISAS.strategies.dingnet_strategy.DingNetStrategy

Figure 3: Upisas Adaption Strategy Class diagram generated in IntelliJ

Figure 3 presented above presents the auto generated class hierarchy form IntelliJ, showing the primitive data types each class implements.

Furthermore, Fig.4 and Fig.5 present the logic described above from the perspective of the activity and sequence diagrams.

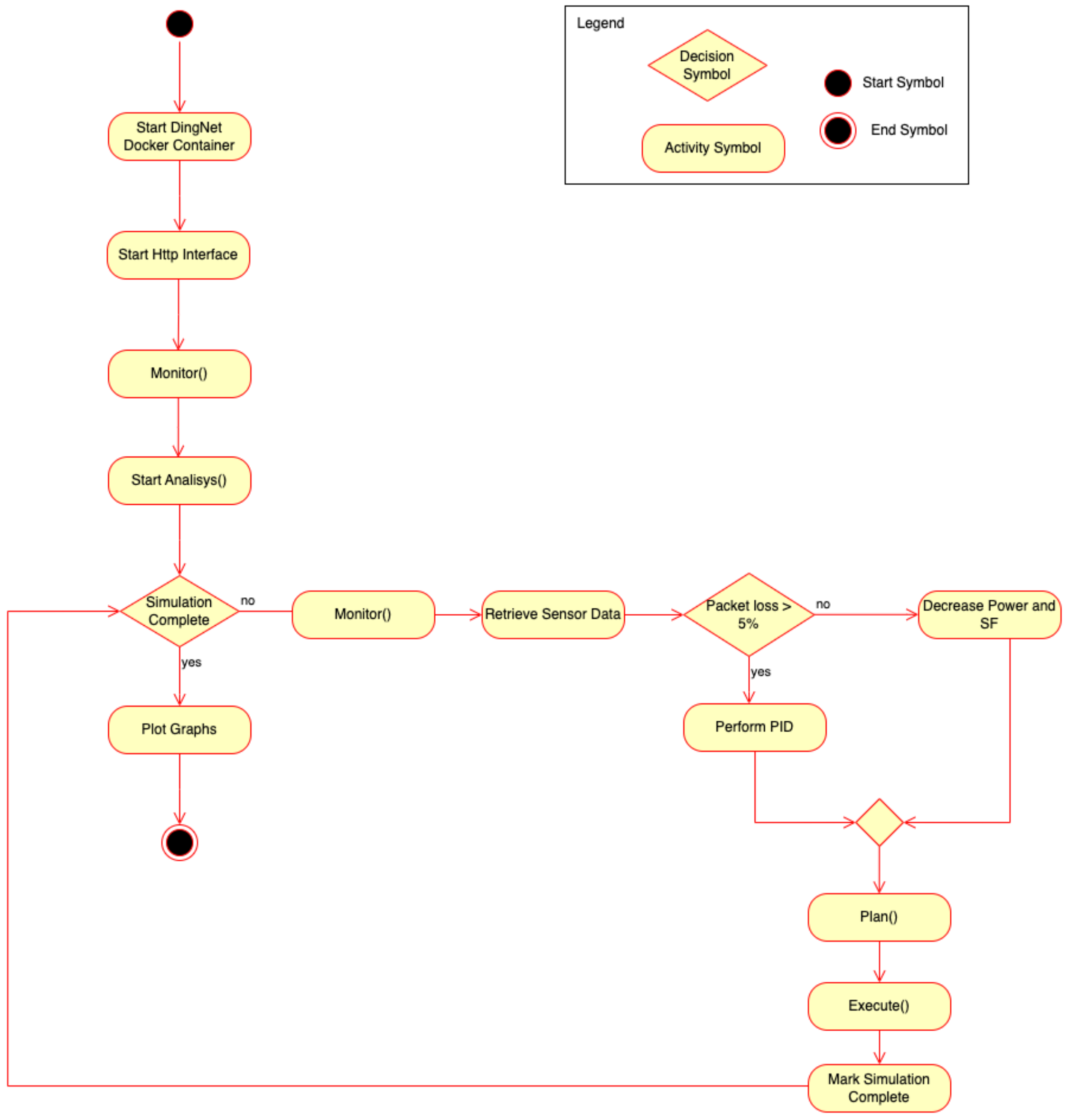


Figure 4: Upisas Adaption Strategy Activity diagram

With regards to the adaption strategy, after the container and HttpServer are started, the system uses a monitor() action to retrieve sensor data for the exemplar to start the analysis. There is a loop which continues as long as the simulation is incomplete, monitoring, retrieving sensor data, and apply the logic for the packet loss threshold described above. Further to these steps, a planning, execution and marking of the simulation as complete are done. After the simulation is finished, graphs are plotted for visual interpretation.

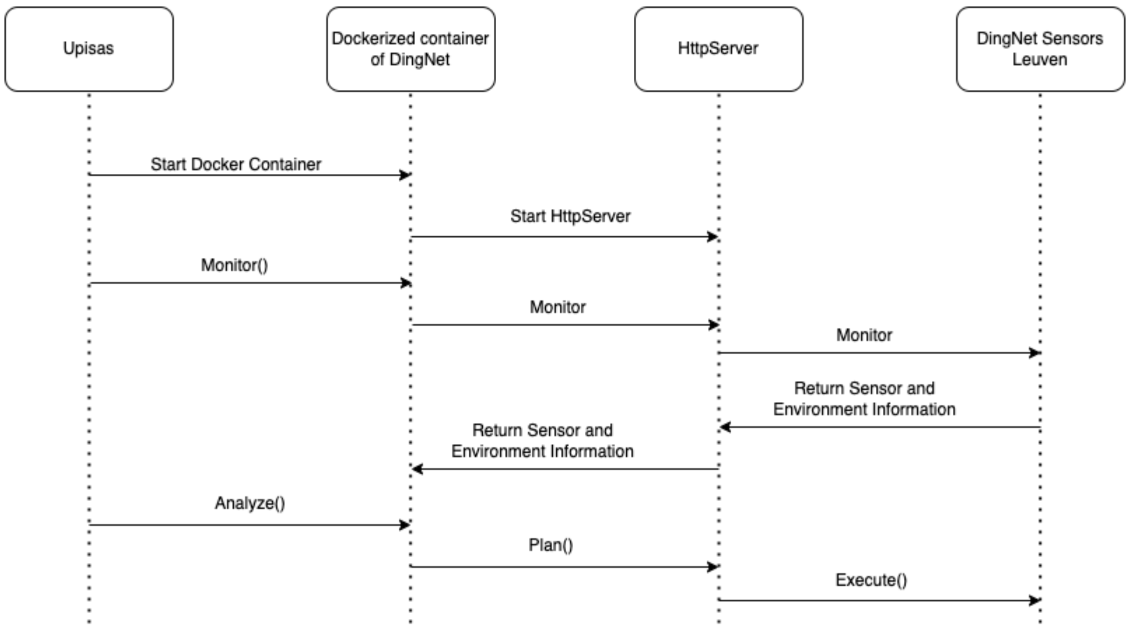
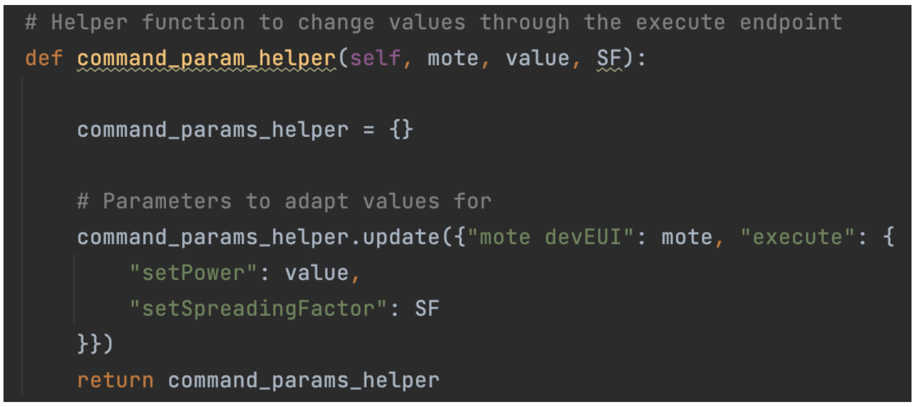
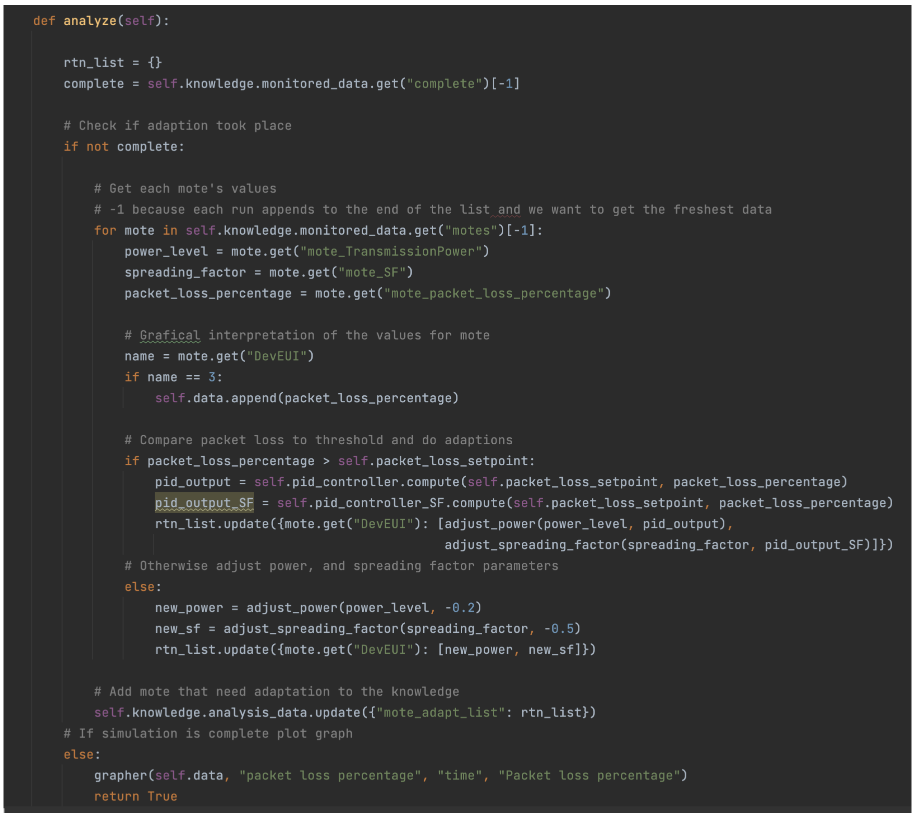


Figure 5: Upisas Adaption Strategy Sequence Diagram



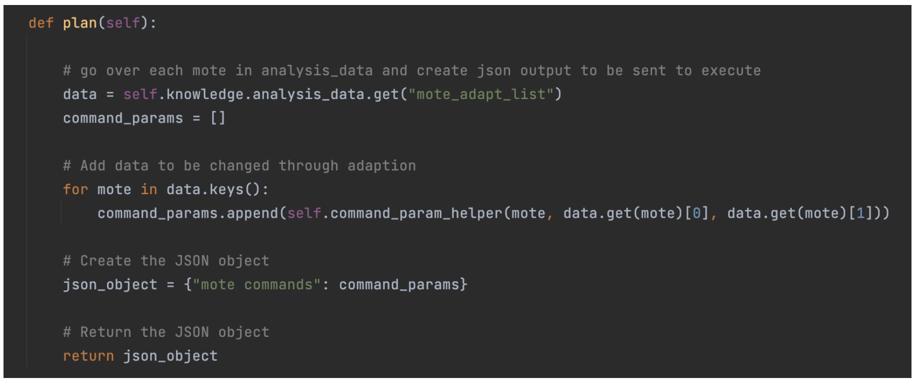
```
# Helper function to change values through the execute endpoint
def command_param_helper(self, mote, value, SF):

    command_params_helper = {}

    # Parameters to adapt values for
    command_params_helper.update({"mote devEUI": mote, "execute": {
        "setPower": value,
        "setSpreadingFactor": SF
    }})
    return command_params_helper
```

Figure 6: command_params_helper() method



```python
def analyze(self):

    rtn_list = {}
    complete = self.knowledge.monitored_data.get("complete")[-1]

    # Check if adaption took place
    if not complete:

        # Get each mote's values
        # -1 because each run appends to the end of the list and we want to get the freshest data
        for mote in self.knowledge.monitored_data.get("motes")[-1]:
            power_level = mote.get("mote_TransmissionPower")
            spreading_factor = mote.get("mote_SF")
            packet_loss_percentage = mote.get("mote_packet_loss_percentage")

            # Grafical interpretation of the values for mote
            name = mote.get("DevEUI")
            if name == 3:
                self.data.append(packet_loss_percentage)

            # Compare packet loss to threshold and do adaptions
            if packet_loss_percentage > self.packet_loss_setpoint:
                pid_output = self.pid_controller.compute(self.packet_loss_setpoint, packet_loss_percentage)
                pid_output_SF = self.pid_controller_SF.compute(self.packet_loss_setpoint, packet_loss_percentage)
                rtn_list.update({mote.get("DevEUI"): [adjust_power(power_level, pid_output),
                                                      adjust_spreading_factor(spreading_factor, pid_output_SF)]})
            # Otherwise adjust power, and spreading factor parameters
            else:
                new_power = adjust_power(power_level, -0.2)
                new_sf = adjust_spreading_factor(spreading_factor, -0.5)
                rtn_list.update({mote.get("DevEUI"): [new_power, new_sf]})

        # Add mote that need adaptation to the knowledge
        self.knowledge.analysis_data.update({"mote_adapt_list": rtn_list})
    # If simulation is complete plot graph
    else:
        grapher(self.data, "packet loss percentage", "time", "Packet loss percentage")
        return True
```

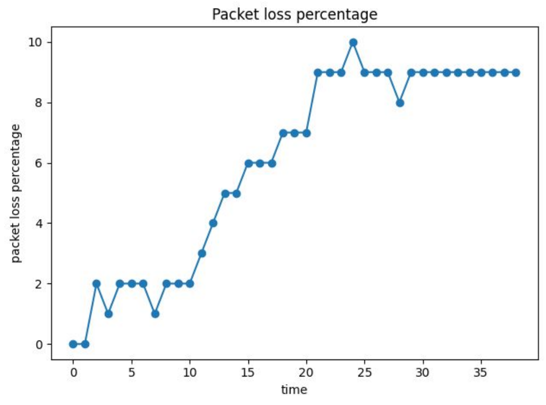
Figure 7: Analyze() method



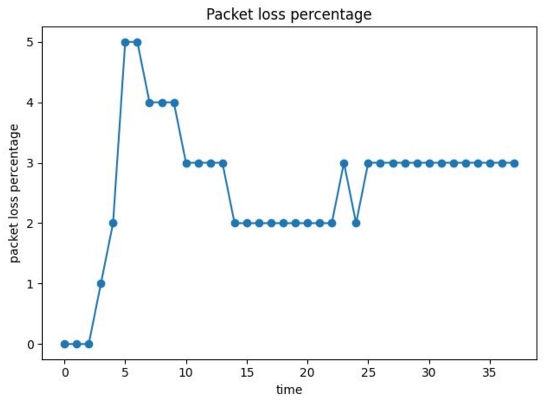
```python
def plan(self):

    # go over each mote in analysis_data and create json output to be sent to execute
    data = self.knowledge.analysis_data.get("mote_adapt_list")
    command_params = []

    # Add data to be changed through adaption
    for mote in data.keys():
        command_params.append(self.command_param_helper(mote, data.get(mote)[0], data.get(mote)[1]))

    # Create the JSON object
    json_object = {"mote commands": command_params}

    # Return the JSON object
    return json_object
```

Figure 8: Plan() method

Regarding the benefits of using the adaption strategy, the results could be seen Fig.9 and Fig.10. When the adaption strategy is applied, the packet loss percentage is reduced, as can be seen in Fig.9b and Fig.10b. For both motes, after adaption the packet loss levels are below 5%, which satisfies the initial assumption. It can also be seen that after encountering the steady state in the

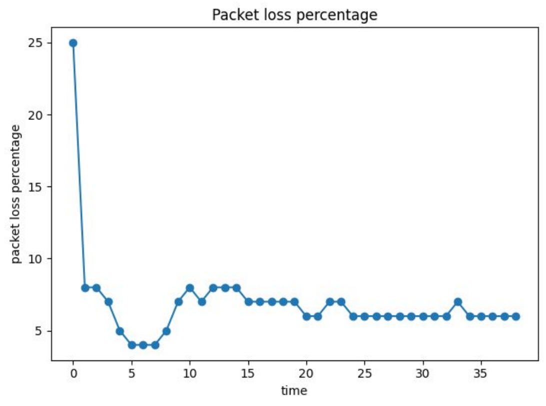


(a) No adaption

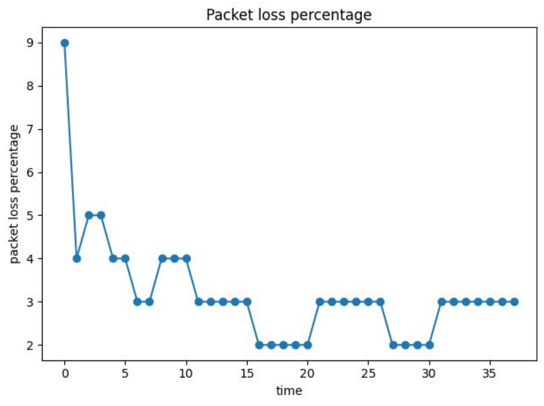


(b) With adaption

Figure 9: Mote 1 values



(a) No adaption



(b) With adaption

Figure 10: Mote 3 values

case with an applied adaption, a small steady state error is present, which may be due to sensor calibration inaccuracies, or different sensor charge levels.

Regarding the side effects of the PID controller implementation, a small steady state error is introduced, as can be seen in Fig.9 and Fig.10 as this type of controller is sensitive to noise and measurements errors, which represent a source of fluctuation amplification, causing oscillations.

6 Implementation

The adaption script is designed to dynamically adjust the transmission parameters of individual network nodes in a LoRaWAN based on real time packet loss data using a PID control strategy. The goal is to optimize the networks performance in terms of reliability and efficiency.

PID Controller: The adaption script defines a PID Controller class implementing a Proportional Integral Derivative (PID) controller. PID controllers are used in control systems to regulate a process by adjusting a control input based on the error, integral of the error and derivative of the error.

Data Visualization: The grapher function is responsible for visualizing data. It uses the Matplotlib library to plot a graph of packet loss percentage over time.

DingNetStrategy Class: This class inherits from a Strategy class and is used to implement a specific strategy for managing and optimizing a wireless communication network.
It initializes PID controllers for adjusting transmission power (pid controller) and spreading factor (pid controller SF), which are critical parameters in LoRaWAN.
The analyze method processes monitored data from motes (network nodes) and decides whether to adjust transmission parameters based on packet loss percentages.
The plan method generates a set of commands to be execute adjusting the transmission power and spreading factor for specific motes.

Helper Functions: adjust power and adjust spreading factor functions adjust the power level and spreading factor, respectively, based on the output of the PID controller. They ensure that the adjusted values remain within operational limits.

Server Startup: The script starts an HTTP server and waits until its up and running.

Main Execution: The script creates an instance of a network exemplar and starts it.
It initializes an instance of the DingNetStrategy class and enters a loop where it continuously monitors the network, analyzes data, executes strategies and sleeps for a short duration between iterations. The loop continues until the simulation is complete at which point it visualizes the packet loss data and exits.

6.1 Technologies

LoRaWAN : The script is designed to control and optimize parameters in a LoRaWAN network. LoRaWAN is a low power, wide area networking protocol designed for long range communication in the Internet of Things applications.

HTTP Server: The script starts an HTTP server to communicate with the network exemplar. The server likely handles requests and responses for controlling and monitoring the simulated or real network.

6.2 Libraries

Matplotlib: Used for data visualization, specifically to plot a graph showing packet loss percentage over time.

UPISAS: The script imports modules from UPISAS, which is custom library. It includes functionalities for getting responses from HTTP GET requests, exemplars for network simulation and a strategy class that the DingNetStrategy class inherits from.

Time: The time module is used for introducing delays in the script.

6.3 Detailed Design

PID Controller Design: The script implements a PID controller for adjusting transmission power and spreading factor. The design includes proportional, integral, and derivative components to make dynamic adjustments based on the error between the desired and measured packet loss percentages.

Strategy Design: The DingNetStrategy class represents a strategy algorithm for managing the network. It includes methods for monitoring, analyzing, and planning adjustments to network parameters.

Adaptive Control: The script employs a feedback control mechanism. When packet loss exceeds a setpoint, it uses the PID controller to dynamically adjust transmission parameters. This adaptive approach aims to maintain optimal network performance.

6.4 Algorithm:

1.PID Algorithm: The Proportional Integral algorithm is a classic control algorithm used to regulate a system by adjusting a control input based on the error, integral of the error .

Proportional (P): This component responds to the current error which is the difference between the desired setpoint and the actual measured value. The proportional term is proportional to this error.

Integral (I): This component considers the cumulative sum of past errors over time. It helps eliminate any long-term steady-state error and brings the system to the setpoint.

Derivative (D): This component considers the rate of change of the error. It anticipates future behavior based on the current rate of change and helps dampen oscillations.

The combined action of these three components is used to compute the control output, which is then applied to the system. The formula for the PID control output is typically expressed as:

$$u(k) = K_p e(k) + K_i \int_0^k e(k)dk + K_d \frac{d}{dk} e(k)$$

(1) Where:

- $\mathrm{u_k}$ represents the control signal at time k.
- $\mathrm{K_p}$ represents the proportional gain.
- $\mathrm{e_k}$ represents the control error measured output value at k minus reference input.
- $\mathrm{K_I}$ represents the integral gain.
- $\mathrm{e_i}$ represents the control error accumulated over time
- $\mathrm{K_d}$ is the derivative gain, and the derivative is taken with respect to time.

2.Adaptation Logic: The script uses logic to decide whether to adjust transmission parameters based on the measured packet loss percentage. If the predefined threshold for packet loss is exceeded, it calculates adjustments using the PID controller otherwise it applies predefined adjustments.

Monitoring: The script continuously monitors the network, collecting information such as the packet loss percentage for each mote (network node).

Comparison to Threshold: The script compares the measured packet loss percentage to a predefined threshold (self.packet_loss_setpoint). If the packet loss exceeds this threshold, it indicates a declination in network performance.

Adjustment Decision: If the packet loss exceeds the threshold, the PID controller is invoked to calculate adjustments. The PID controller computes a control output based on the difference between the setpoint and the measured value, the cumulative sum of errors and the rate of change of error.

Parameter Adjustment: The calculated PID output is used to adjust specific transmission parameters, such as transmission power and spreading factor, for the affected motes.

Predefined Adjustments: If the packet loss is below the 5% threshold, energy usage is optimized by incrementally decreasing each mote's power level by 0.2 and spreading factor by 0.5.

6.5 Pseudo code:

pseudo code is on the main execution block of the script. It starts the network exemplar, initializes the strategy and enters a continuous loop where it monitors the network, analyzes data, executes strategies and introduces a delay between iterations. The loop continues until the simulation is complete.

```
main():
    exemplar = dingnet.dingnet(auto_start=True)
    _start_server_and_wait_until_is_up(exemplar)

    strategy = EmptyStrategy(exemplar)
    while True:
        strategy.monitor()
        complete = strategy.analyze()
        if complete:
            break
        strategy.execute(strategy.plan())
        time.sleep(0.8)
```

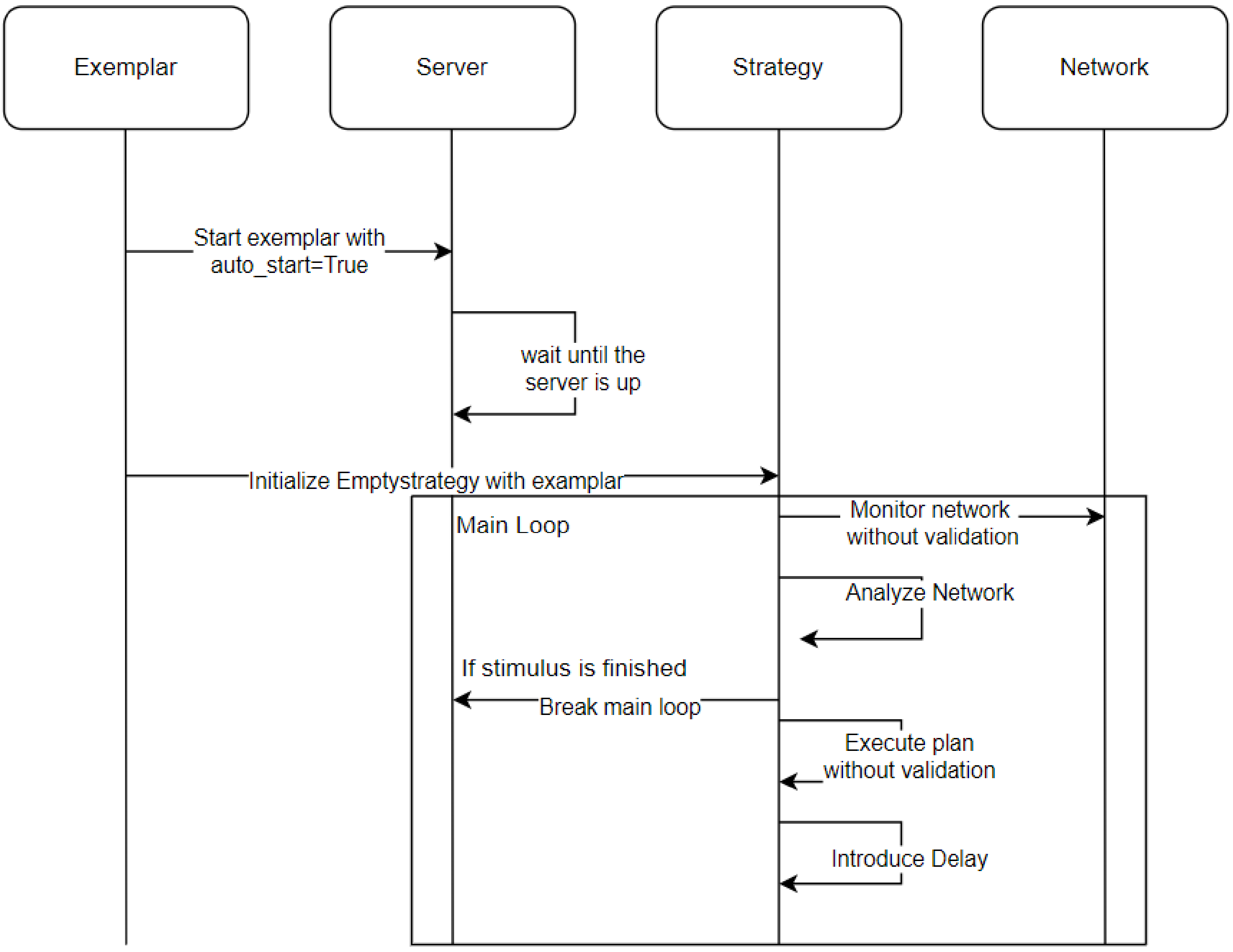

Exemplar
Server
Strategy
Network
Start exemplar with auto_start=True
wait until the server is up
Initialize Emptystrategy with examplar
Main Loop
Monitor network without validation
Analyze Network
If stimulus is finished
Break main loop
Execute plan without validation
Introduce Delay

Figure 11: sequence Diagram

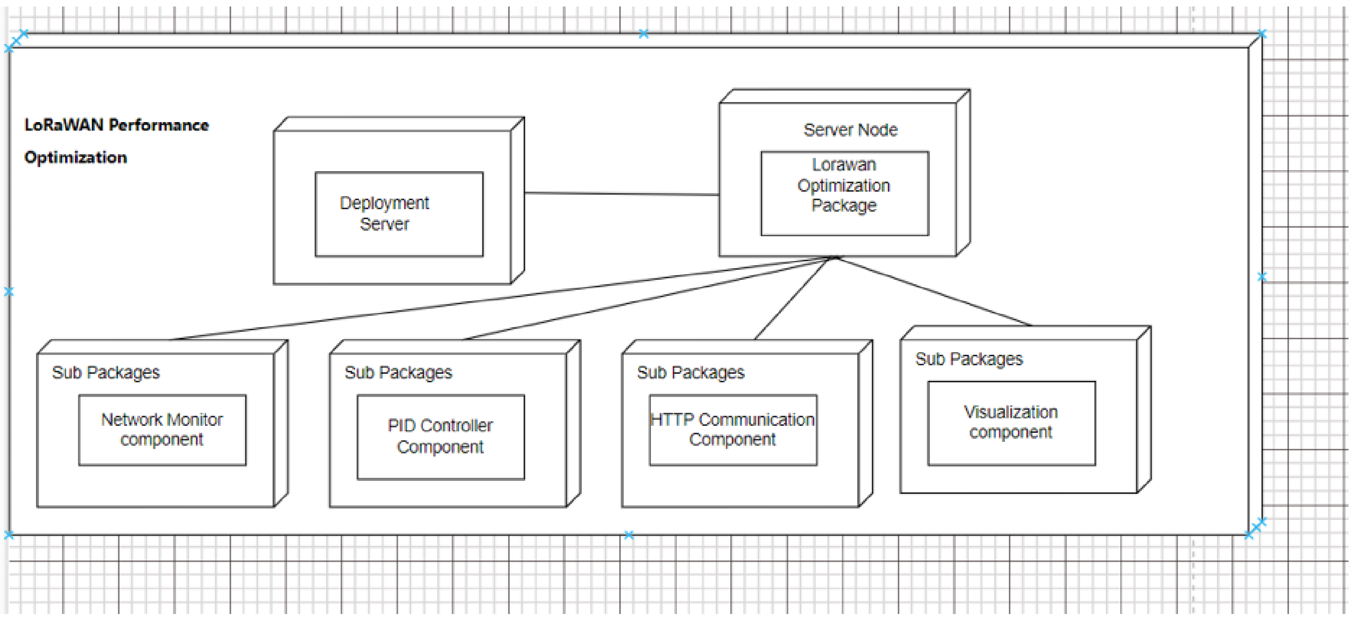


Figure 12: Deployement Diagram

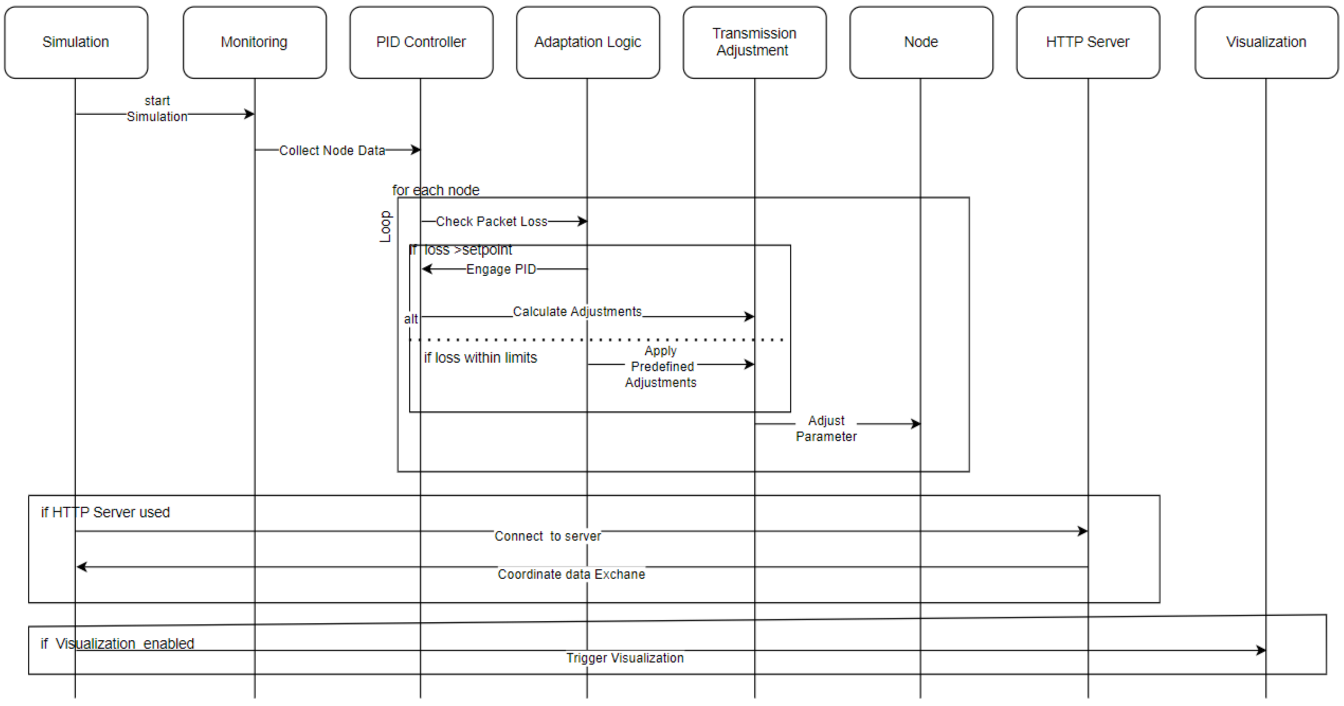


Figure 13: Implementation Process sequence Diagram

7 Showcase and evaluation

The goal of this self adaptation project, as described in chapter 2, is to keep packet loss between motes and gateways under a threshold and optimising energy consumption in the motes. Figure 9 and figure 10 shows the packet loss percentage of each mote before and after the adaptation has been applied in a single run. Hence, this offers a good approach to evaluating how the adaptation strategy improves on the baseline.

Figure 9a and 10a shows the packet loss of mote 1 and mote 3 during transmission respectfully. It can be seen that the packet loss for mote 1 reaches a peak of approximately 9%, and mote 3 reaches a peak of 7% under normal operation. The packet loss on mote 1 and 3 after the implementation of the PID controller adaptation strategy, to keep packet loss below 5%, can be seen in figures 9b and 10b. It is seen that mote 1 and 3 now settles at a packet loss of 3%. Hence, it can be concluded that the adaption strategy is effective in keeping packet loss below 5%.

Looking closer at figures 9b and 10b it can be seen that the settling time is 15 seconds after which the packet loss reaches the steady state phase seen by the fluctuating graph. The steady state error could be caused by the PID controller values but could also be a side effect of decreasing the energy consumption by reducing the transmission power and spreading factor variables which result in an increase in packet loss, which the PID controller will then try to reduce. Thus causing a cyclical pattern of packet loss.

An advantage of our adaption strategy is that there is no need for runs dedicated to learning, like in q-learning, The adaptation can begin immediately. This reduces reduces the time before the adaptation is noticeable in the operation of the exemplar. However, a downside is that the managing system is executing changes on the exemplar. That is, every second the managing system is making a change to the transmission power and spreading factor of each mote. This is a result of how the PID controller works. It is possible that these interaction cause an overhead in the exemplar. A different adaption strategy could simply monitor the exemplar and makes changes when necessary. Thus reducing the amount of execute calls in the exemplar and reducing overhead.

Chapter 9 contains a link to a video that shows how the adaptation strategy works.

8 Reflection

Working on the DingNet project offered valuable insights into the adaptive software for IoT systems in smart cities. We navigated the complexities of applying theoretical approaches to real-world scenarios, grappling with the nuances of self-adaptation and facing the technical challenges of integrating real-time data. This reflection delves into our learning experiences, the challenges faced, and the future possibilities for DingNet's evolution in the smart city landscape.

- **Understanding of Adaptive Software and IoT Systems:** Working on the assignment provided a comprehensive understanding of the complexities involved in developing adaptive software, particularly in the context of Internet-of-Things (IoT) systems. The DingNet exemplar was instrumental in illustrating practical applications in smart city contexts.
- **Importance of Self-Adaptation:** The assignment also helped us to understand the critical role of self-adaptation in managing the complexities and uncertainties inherent in modern software systems, especially those dealing with Internet Of Things domain.
- **Challenges in Implementation:** In implementing self-adaptation concepts, a significant challenge was developing an effective adaptation strategy. The decision between using Q-learning and PID (Proportional-Integral-Derivative) controllers required careful consideration.

These choices necessitated deep understanding of both theoretical concepts and practical implications. Additionally, an attempt was made to integrate a weather API for real-time weather updates in the DingNet simulator. It was aimed at enhancing the system's responsiveness to environmental changes. However, this integration faced technical hurdles, and despite efforts, the API could not be effectively utilized in the project. This experience highlighted the complexities involved in applying theoretical concepts to real-world scenarios, especially when adapting to dynamic environmental factors. Integrating the theoretical knowledge from the course with the practical application in the DingNet simulator was a complex but rewarding process. It required a balance of understanding the principles of adaptive software and applying them in a simulated environment.

- **Learning through Experimentation:** The hands-on experience with the DingNet simulator was indeed a critical aspect of learning. Experimenting with different scenarios and adaptation strategies in the simulator was both enlightening and challenging. Notably, a significant portion of this experimentation involved determining the optimal PID (Proportional-Integral-Derivative) controller parameters. This required conducting numerous trials, adjusting and fine-tuning these parameters repeatedly to achieve the desired system behavior. Such iterative experimentation underscored the importance of practical application in understanding and implementing complex concepts like self-adaptation in software systems.

- **Future Scope :** The future of DingNet in adaptive software for IoT and smart cities includes enhancing learning algorithms for better self-adaptation, integrating diverse real-time data sources like traffic and pollution sensors, improving scalability and efficiency for larger networks, developing user-centric adaptations for personalized experiences, and focusing on robustness and reliability to ensure consistent performance under varying conditions. This direction offers a rich ground for research and development, pushing the boundaries of IoT applications in smart cities.

9 Links to private Github repositories & Video demonstrating adaptation strategy

The repositories we have worked on can be found here:

- `https://github.com/CristianSusanu/DingNet_Group_5_1/tree/main`
- `https://github.com/CristianSusanu/UPISAS_Group_5_1/tree/main`
- `https://youtu.be/HdqwFtZRYAg`

9.1 How to run the code

1. Clone the latest version of the dingnet and upisas repositories. Make sure you pull the main branch.
2. Run the dockerfile located in the dingnet project. For this you should have docker installed and working. Run this from the commandline inside the dingnet project : " `docker build -t dingnet .` " .
3. In the upisas project run: " `pip install -r requirements.txt` ". This will ensure you have all the python requirements. It is recommended to do this inside a python virtual environment.
4. To run the adaption strategy do: " `python dingnet_strategy.py` ". You may need to replace `python` with `python3` .

10 Division of work

Name	Individual contribution summaries
Sree Padmavathy	• Completed the overall system description for the adaptive system, including a general overview and integration details (Section 1) • Analyzed uncertainties within the system and detailed adaptation strategies to address them. (Section 2) • Translated traditional requirements into adaptive terms using natural and RELAX language constructs. (Section 3) • Explored potential solutions, providing analysis on the adaptation logic of the system. (Section 4) • Worked on the reflection section on key takeaways and challenges encountered during the project.(Section 8)
Dalvie	• Peer programming sessions for DingNet and Upisas • Section 7 of the report.
Cristian	• Peer programming sessions for DingNet and Upisas • Section 5 of the report.
Yuktha Sri	• Section 6 of the report

References

[1] M. Provoost and D. Weyns, “Dingnet: A self-adaptive internet-of-things exemplar,” in *2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, IEEE, Leuven, Belgium, 2019.

[2] J. Whittle, P. Sawyer, N. Bencomo, and B. H. Cheng, “A language for self-adaptive system requirements,” *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2008.