Contents

Todo list

Chapter 1

Introduction

The case chosen for this assignment was project 3 regarding simulator for wireless communication. Based on the assignment description, the report will contain information about all the phases of the project; Analysis, Design, how the design was implemented, major decisions that we needed to take and finally the live product.

The wireless network simulator involved designing a software that simulates wireless communication between nodes. Furthermore, the nodes are to be places in different position in a user defined space and they are to communicate with each other. In the assignment description, the program is required to have these additional aspects:

- User must choose different type of nodes such as base stations, mobile phones, drones, satellites...

- A subset of nodes may transmit a block of data selecting a certain power, frequency, bandwidth and time of transmission

- A subset of nodes may listen to incoming data and try to decode the data

- A GUI is used to insert and remove nodes. It is also used to start and stop data communication and to show communication quality information of each node

We then proceeded to carry on with the first stage of the waterfall model which is Analysis. During the Analysis, we discovered additional considerations that we would need to take into account. This resulted in the group coming up with a use-case diagram and a class diagram. Then the simulator was implemented in python. During this project, the group had encountered some challenges which would be discussed in the report.

Chapter 2

Analysis

In the first step of the waterfall model, requirement and object analysis is carried out.

2.1 Defining use cases

Based on the description given, actors and use cases are identified and then clearly defined.

2.1.1 Actors

Based on the project description provided, the only actor that we have found would be the user who is running the simulation. At first we had considered the database to be an actor of the system but later discovered that it would not be that appropriate to have the database as an actor to the system.

2.1.2 Use cases

Next, we have defined the various use cases that we want the user to be able to do, which will be described in the following sections.

Add node

When the user wants to add the node, they would need to specify the type of node added. The node added could be either a IoT Node or a network gateway.

Start Simulation

When the user wants to start the simulation, they would need to provide the following information:

- Number of nodes in the network

- Size of the 2D grid where the nodes will be placed

- Option of verbose output

See simulation statistics

When the user wants to see simulation statistics, the system would need to check if any simulation has occurred previously. If there is no prior simulation, then the system would display an error. Otherwise, the system would display the simulation statistics such as throughput, delay, and packet loss.

2.1.3 Use case diagram

Based on the factors mentioned above, a use case diagram was developed. The use case diagram can be seen below.

2.1.4 FURPS

The FURPS were utilized to better define the system's requirements. The system's overall summary was also created using this technique. This approach also aided in providing a fresh and more enlightening perspective on the issue and the requirements.

Functionality: what should the system do?

- The system should be able to place the nodes in a grid and transmit packets from one node to the gateway.

- The system would need to run the simulation for a fixed amount of time

Usability: what kind of UI is needed?

- A screen would be required to display the simulation statistics

- A device should receive user input eg.keyboard

Reliability: what is the tolerance of the system to failures?

- The system should be able to check if the simulation has been run before displaying simulation statistics

Performance: which response times, accuracy, availability, resource usage should the system have?

- The GUI needs to be run on a simple computer

- The simulation statistics needs to be stored in a server

Sustainability: what adaptations may be needed?

- The system need to give the user the ability to change the simulation conditions after a run has been completed

2.2 Domain model

To aid the group in identifying the various objects involved in the problem, a domain model was developed. Firstly, the identified actor, which is the user, is represented as a GUI in the model. This also constitutes the front end of the system. Regarding the back end of the system, we would need one object to simulate the network and another one to generate/store the simulation statistics. Ultimately, we would need an object for the devices themselves and one for the overall network. Here it might also be a viable option to differentiate between a node object and a gateway object that then both can inherit attributes and methods from the device object when making the class diagram. Then, in order to control everything and tie the simulation to gather, it would make sense to incorporate a simulation object that handles the simulation itself.
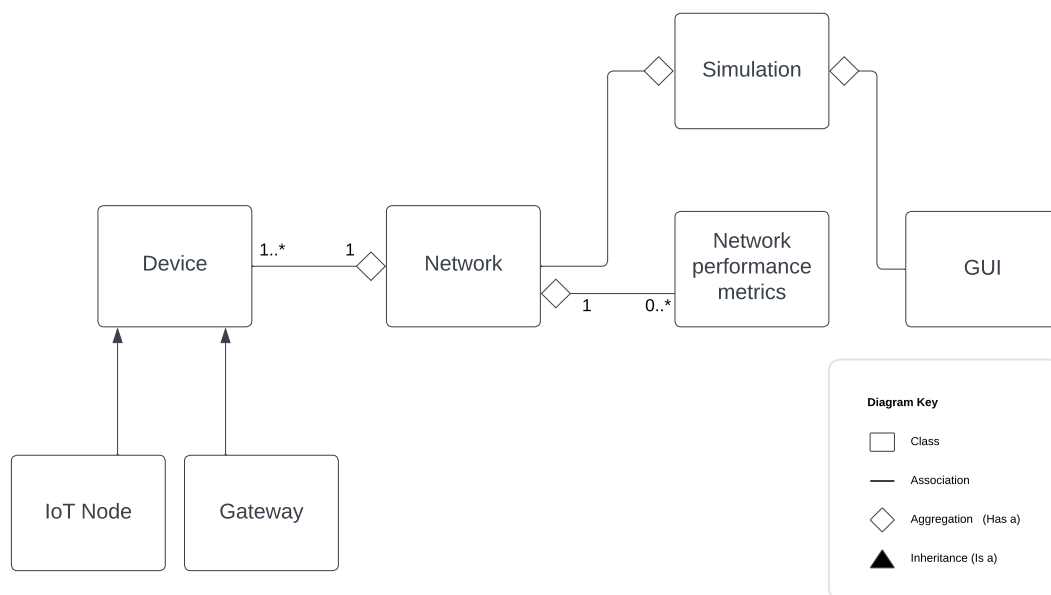

Figure 2.1: Diagram of the domain model.

2.3 Class diagram

The entirety of the class diagram can be seen in Figure 2.2. The parts that were not implemented have their text colored red. This also implies that the class diagram is made in accordance to how the implementation has been done, where it then has been changed whenever something has been changed in the program itself.
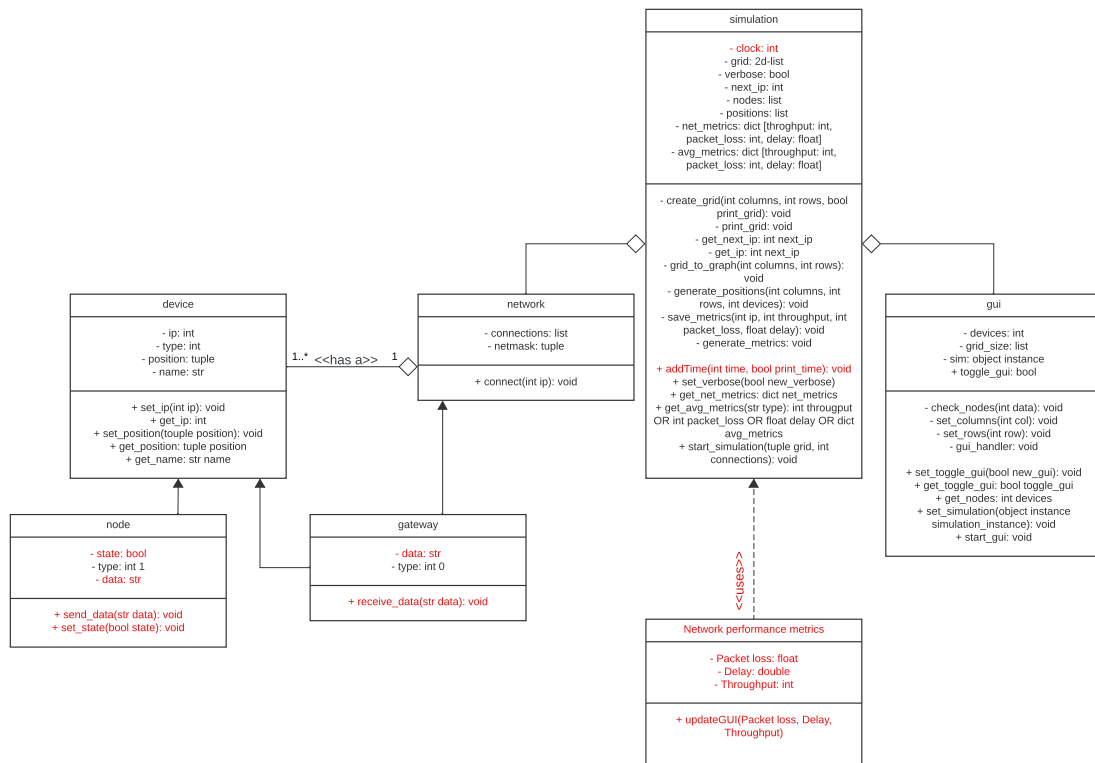
Figure 2.2: Diagram of the classes.

2.3.1 GUI

It needs to be possible for the rest of the system to function without a GUI, which will also make it easier to test whether the rest of the program functions as expected. Since this is also the object that the user will interact with, it should also be somewhat easy to use. In order to start the GUI itself a `start_gui` method is needed that can then be called from the main program file. Since PyWebIO is used to program the GUI, the built-in method, called `start_server` can then be called to open a port on the transport layer, which then in turn can be used to access the GUI from any other computer with network access to the computer executing the program. When `start_server` is called, another method can be passed along as an argument, which is then how the GUI is controlled for the remaining time. This method is called `gui_handler`, and it decides which elements of the GUI is shown as well as receiving input from the user and calling other methods of the `gui` object.

2.3.2 Network

This class is missing a lot of the attributes and methods that it should have managed instead of the `simulation` object. As it is currently implemented, the `network` object only stores a list of the IP-addresses that are currently reserved by the gateway or one of the nodes.

Conventionally, on local networks, only the digits after the last period in the IP-addresses differentiate the different devices in the network. Therefore, it is only necessary to store

the differentiating part of the IP-addresses. This means that the attribute `connections` must be able to contain a list of integers from 1 to some maximum number of devices in the network. Additionally, the `simulation` ended up being in charge of creating connections between the gateway and the nodes, even though this should definitely have been a method of the `device` object that `node` and `gateway` would then have been able to use to reserve an IP-address in the `connections` attribute of the `network` object.

To put it differently, the `network` object should have more coupling to make it more clear that it is in charge of the network functionality, that its name is implying. Even though the `simulation` is necessary, it should have been possible to view all of the network methods and attributes in the `network` object instead of needing to also view the `simulation` object.

2.3.3 Device

As the class diagram in Figure 2.2 shows, `device` is a superclass of `node` and `device`. This is done to avoid duplication of the methods and attributes that `node` and `gateway` share. The attribute `type` is set to `None` in the `device` object. This is done since this attribute is not intended to be used in an instance of the `device` object, since the `set_ip` method takes the `type` and `ip` attributes and set the name of the device (either a `gateway` or `node`). The specific naming of devices will be further specified in subsubsection 2.3.3 and subsubsection 2.3.3. In the final implementation, the `position` also ended up being used to store the position of a given node in itself. However, the device does not need to know its own position - only the `simulation` object should be aware of the position of each device since the `position` is only used to print the position in the terminal in the current implementation. The multiplicity between `network` and `device` is that one network can have one to many devices. However, as of the current implementation, there can only be one gateway.

Node

This subclass of the `device` overwrites the `type` attribute to 1, when an instance of it is created. The naming of each `node` instance is then done based on their `ip` attribute. The gateway uses the IP-address 1. This means that the first node has the IP-address 2. The naming of each `node` is therefore 'Node n', where n is 2 subtracted from the `ip` attribute. The attributes and methods with red text will be further touched upon in chapter 3, where the implementation will be discussed.

Gateway

In the final implementation, the `gateway` ended up being a subclass of both the `network` and the `device`. Just like the `node` class, the `gateway` overwrites the `type` attribute of the `device` class. The naming in `set_ip` for a `gateway` with `type = 0` is then 'Gateway n', where n is 1 subtracted from the `ip` attribute, since the first and only gateway has the IP-address 1.

2.3.4 Simulation

The simulation is the object that takes over right after getting called by the `gui` object. It is then in charge of controlling the simulation. The first thing that happens is then that

the `start_simulation` method is accessed by the `gui_handler` method which passes on the input from the user, namely the attributes `devices`, and `grid_size`. By called its own method `simulation` can then use `create_grid` to create an empty grid of the size specified by the previously mentioned attributes.
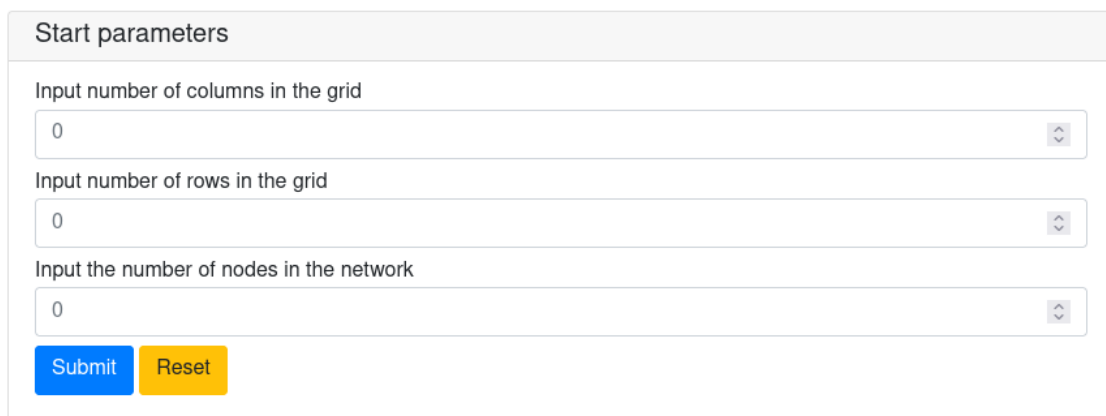
Right after creating the empty grid, the `gateway` object is instantiated, it gets the IP-address 1 by using calling the `get_next_ip` method, and its position is then set to the middle of the empty grid and stored both in the `gateway` object as well as the `simulation` object. Even though it was implemented this way it should only have been stored in the `gateway` object.

Chapter 3

Discussion

3.1 GUI

When a user of the system lay their eyes on the GUI, the first elements, they will see, are inputs for how many columns and rows should be in the grid that makes up the system as can be seen in Figure 3.1. Even though the user might be quite knowledgeable about networking, the will only be able to guess what this grid is. Therefore, an easy to implement solution could be to provide a simple description of the use case of the system. An even better solution could be to provide an image that changed along with the users input, and then provide some visualisation of the size of the grid such as axes to clarify even further that the user is changing the graph or grid that will make up the area where the devices in the network simulation can occupy.

Figure 3.1: A screenshot of the first elements that are displayed when a user opens the GUI.

Then, after the simulation has finished running, average metrics of the results from the individual nodes are displayed printed as is shown in Figure 3.2. An improvement to this implementation might incorporate an option to view the results from each individual node by either having the results in a simple table or implementing some way to display the results by having the user click on a button with the name of the node (i.e. Node n results). The option of using an element that is already present would be to enable the network topology graph to function as buttons, where the user would be able to click on the number of a given node and then have the results displayed. An example of the

network topolgy graph can be seen in

Average throughput: 24.2 MB/s

Average packet loss: 5.60 %

Average delay: 12.04 ms

Figure 3.2: A screenshot of how the results of the simulation is outputted in GUI.

3.2　Network

As has already been described in subsection 2.3.2 and in the rest of section 2.3, a better implementation of how the system currently is implemented would have tried to get as close to functional cohesion as possible in order clarify and improve which object does what. As the implementation currently is, a name that more precisely describes the functionality of the `simulation` object would be `simulation_and_networking`. Of course, it must be acknowledged that complete functional cohesion will not make sense in this project, where

3.3　Device

3.3.1　Node

3.3.2　Gateway

3.4　Simulation