UNIVERSITY
OF TWENTE.

April 21, 2024

Design Project Report
Digital Twin for Swarm Applications

Bogdan Colţa
(s2689014)

Dan Negru
(s2683628)

Mihai Spinei
(s2759969)

Nathan Jongejan
(s2690314)

Yuliya Alyoshyna
(s2730669)

Supervised by: Yanqiu Huang

Abstract

Research on swarm applications involving drones in the domain of distributed applications is currently limited. The objective of this design project is to develop a digital twin for swarm applications. Achieving this goal requires the integration of pre-existing components from Thales: the Distributed Simulator for MaritimeManet (DSM) and the Batman Topology Configurator (BTC). Additionally, modifications to the DSM are necessary for enhancing its usability and a new user interface has to be implemented for facilitating the process of creating topology configurations and visualizing their respective simulations.

This report serves as a guide, describing the integration process, the improvements made to the DSM components, the new user interface, and offering recommendations for future work in this project.

Contents

1. Introduction

1.1. Company's background

Thales Group is a multinational company that specializes in providing solutions and services in the fields of aerospace, defence, security, and transportation. They offer a wide range of products and services, including avionics, satellite communication systems, cybersecurity solutions, transportation systems, and digital identity and security solutions. Thales serves a diverse customer base, including governments, military organizations, airlines, transportation operators, and critical infrastructure providers.

1.2. Domain description

In distributed mission systems, various platforms collaborate to execute missions, each tasked with specific responsibilities. The effectiveness of this collaboration relies heavily on the seamless exchange of information among the platforms. This information exchange is facilitated by a collection of processes, forming a distributed application, which operates across the platforms. In maritime scenarios, these platforms include large ships, small boats, unmanned vehicles (both surface and aerial), fixed maritime structures like oil rigs, and occasionally commercial ships. Together, they constitute a distributed mission system capable of operating across vast distances, spanning several kilometres. However, communication between these platforms poses a challenge due to the large maritime environment and the constant movement of the sea.

1.3. Assignment description

MaritimeManet represents a novel ad-hoc wireless network specifically designed for distributed mission systems in naval environments. It addresses the challenges of communication in maritime settings by making use of multiple directional antennas arranged in a sunflower pattern, known as a Multi-Beam Antenna (MBA). This unique feature enables long-range transmission with moderate power levels. Additionally, MaritimeManet incorporates a patented Distributed Neighbourhood Discovery (DND) process, ensuring the establishment of the strongest possible wireless connections between directional antennas of nodes. As platforms move, causing changes in topology and wireless links, the system adapts dynamically to maintain connectivity.

The assignment involves the integration of MaritimeManet components, including the Distributed Simulator for MaritimeManet (DSM) and the Batman Topology Configurator (BTC), to create a laboratory environment for testing distributed applications. The DSM simulates the behaviour of MaritimeManet nodes and processes, while the BTC mimics the multiple directional antennas and allows for topology changes. By integrating these platforms, users can test the behaviour of distributed applications in a controlled environment that accurately replicates the challenges of real-world maritime communication networks. The objectives of the project can be found in Section 2.2.

1.4. Report structure

This report provides a detailed overview of the project process, from planning to results. It begins with task decomposition and prioritization in Section 2, followed by specification and prioritization of requirements in Section 3. A short risk analysis is conducted in Section 4. The overall architecture of the system is elaborated upon in Section 5, describing the design of the new components and their integration with the existing DSM and BTC. The results of the integration and the user interface are detailed in Section 6. Testing strategy, approach, and results are outlined in Section 7. Suggestions for future work are provided in Section 8, while the group reflection and task division are addressed in Section 9. Finally, the report ends with a conclusion in Section 10.

1.5. Components description

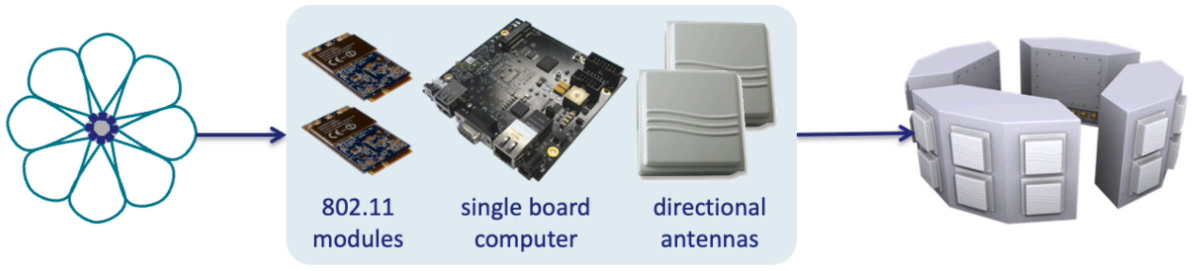


Figure 1: Multibeam Antenna

- **Distributed Mission Systems (DMS)**: Refers to multiple platforms working collectively to execute missions, where tasks are decomposed and allocated to individual platforms. Collaboration among these platforms relies heavily on information exchange.
- **Distributed Application:** A collection of processes running on platforms in a DMS, responsible for exchanging information between platforms.
- **MaritimeManet**: ad-hoc self-organizing wireless network for radio communication. It is designed specifically for distributed mission systems in naval environments. The MaritimeManet node uses several beams arranged in a sunflower pattern as shown in Figure 1. In MaritimeManet, the main idea is to automatically find other nodes and create the strongest wireless connection possible between them. It does this by picking the best antennas at each node. As nodes move around, MaritimeManet keeps checking and adjusting connections to maintain the best possible link. So, if a node moves, it might switch to a different antenna or make new connections as needed.
- **Distributed Neighbourhood Discovery (DND)**: A patented process in MaritimeManet responsible for establishing a connected network with the strongest possible wireless links between directional antennas of nodes.
- **Distributed Simulator for MaritimeManet (DSM)**: A digital twin of MaritimeManet designed for testing its functionality and improvements in a simulated environment.
- **Batman Advanced**: A routing protocol chosen for MaritimeManet to facilitate multihop forwarding of data units in the network.
- **Batman Topology Configurator (BTC)**: A tool used for testing the correct operation of Batman on MaritimeManet by mimicking multiple directional antennas and enabling topology changes in the network.

2. Approach

2.1. Project management approach

For the planning of the project, the team will utilize the waterfall method which is a project management approach that has a linear progression. The reason for using the waterfall method is that the requirements are fixed in the first iteration, which aligns well with the nature of the project. This approach is particularly suitable for this project since the scope is well-defined, and changes to requirements are expected to be minimal. The project will be divided into the following phases:

1. Requirements Analysis
2. System Design
3. Implementation
4. Testing
5. Documentation
6. Deployment
7. Evaluation

2.2. Scope of the project

- Integrating the DSM and BTC application is our primary goal for the project with the utmost priority.
- Similarly is developing a method for the BTC to dynamically configure the topology based on the DND output, simulating changes in the wireless network.
- An additional goal is to allow the integrated system to simulate more complex node paths and movements, which would allow more detailed testing of the system.
- The current solution for presenting the output of the system should be improved and extended to include additional features, such as monitoring the network conditions.
- Improve the capabilities of the input interface to allow for easier and more intuitive configuration and visualization page.
- Testing the resulting integrated system with an example application defined by THALES
- Conduct thorough testing to ensure that the integrated DSM-BTC system accurately reflects the behaviour of a real swarm in terms of performance (delay, bandwidth, loss).
- Document the integration process, including any modifications made to DSM and BTC.
- Improve the maintainability of the project by providing user documentation for operating and configuring the integrated system.

2.3. Communication with the client

The team maintains a weekly meeting every Wednesday with the client. Ahead of each meeting, a progress report is submitted to the client, providing an overview of work done and problems faced. During the hour-and-a-half session, the team engages in interactive discussions, addressing queries, presenting progress, and showcasing completed work. Additionally, a Microsoft Teams channel ensures continuous communication, allowing the client to share resources and communicate changes outside the scheduled meetings.

3. Requirements Specification

3.1. Requirements capturing

The requirements for the integration task were discussed and verified with the client. It was necessary to elicit and discuss the requirements at the start of the project, as we utilized the waterfall project management approach. Nevertheless, requirements regarding configuration and visualization were covered later in the project; the client obtained these requirements through communication with end users, which they then shared with us. The configuration and visualization requirements took a more iterative approach in order to be flexible and find a variety of solutions.

3.2. Stakeholders

We would categorize all the following stakeholders as ***end-users*** of the DSM and BTC:

- Naval Operators
- Maritime Engineers
- Disaster Response Teams
- Researchers

3.3. Functional requirements

The following requirements are listed and prioritised using the **MoSCoW** method:
M — Must have — non-negotiable needs for the project.
S — Should have — essential to the product, project, or release, but they are not vital.
C — Could have — nice features to have.
W — Won't have — features that fall outside the scope of our project.

1	**Integration**	**MSCW**
1.1	The DSM and BTC should be able to establish a connection between each other	M
1.2	The DSM should be able to communicate changes in its topology to the BTC	M
1.3	The DSM should be able to communicate link properties to the BTC	S
1.4	The BTC should be able to simulate bandwidth, delay, and packet loss	S
1.5	The BTC should simulate given link properties per link	S
1.6	The DSM should be able to track changes in topology and store them	M
1.7	The BTC should be able to realize any possible topology	M
1.8	The BTC should be able to connect/disconnect any 2 nodes virtually	M
1.9	Two directly unconnected nodes should not be able to communicate directly with each other	M
1.10	If the topology changes, two nodes which stay connected should be able to continue to communicate without interruption	S
2	**Configuration and Visualization**	
2.1	The route configurator should enable the user to specify a node's trajectory, either via waypoints or drawn line	S
2.2	The route configurator should support a split-join movement in the simulation	M
2.3	The route configurator should be able to save a movement sequence to a file	M

2.4	The input configurator should allow the user to drag and drop nodes to specify their initial location	M
2.5	The input configurator should allow the user to configure the node's settings such as type of antenna used and translation/rotation speed	M
2.6	The input configurator should allow the user to select a node by clicking on it for further modifications	M
2.7	The input configurator should allow the user to add/delete a node by clicking a button	M
3	**Output presentation**	
3.1	The DSM should be able to present the current topology in a GUI	S
3.2	The BTC should be able to capture the routing table configuration and present it in the GUI	C
4	**Virtualization**	
4.1	The BTC should allow the configuration of a virtualized Batman environment	C
4.2	The current Batman environment should be replicated in a virtualized environment	C

3.4. Non-functional requirements

The output presentation of a topology should be user-friendly
The configuration and visualization of topology should be intuitive and graphically based, allowing the user to specify a node's position via drag and drop, configure a node's trajectory graphically, etc.
The system should have low latency, ensuring that the exchange of information between DSM and BTC components occurs with a response time (time taken to send a message and receive a response) not exceeding 1 second
The BTC should be extensible to allow further development on top of our test bed

4. Risk Analysis

1. **Requirements stability:**

Since the waterfall methodology will be used as a project management tool, the requirements are fixed in the beginning; to mitigate this risk, we will conduct thorough requirements analysis and verify them with the client.

2. **Technical stability:**

Integrating components and improving the UI might have unforeseen technical challenges; to mitigate this risk, we maintain flexibility in our development phases.

3. **Graphical solution stability:**

The graphical solutions chosen to be investigated at the beginning of the project might not be suitable; conduct thorough evaluation and prototyping of graphical solutions before integration; have a fallback plan for an alternative solution.

4. **Timeline constraints:**

Unforeseen delays may occur; to mitigate this risk we will have daily meetings, where our team plans what will be done for the day and check in regularly on the progress.

5. **Communication stability:**

Ineffective communication and misunderstandings; to mitigate this risk we will have regular meetings with the client each week, biweekly meetings with the supervisor and daily meetings with each other to ensure that everyone is up-to-date with the development process.

5. System Architecture

5.1. Introduction

This section will describe the functional architecture of the integration between the Distributed Simulator for Maritime-Manet and the Batman Topology Configurator, as well as the additional components developed to interact with the system and the architectural changes required for this integration. We will start with an overview of the whole system and compare it to the initial structure. We will describe each component, its responsibilities and the data interfaces used for communication, as well as the flow of data through them.

5.2. Initial design

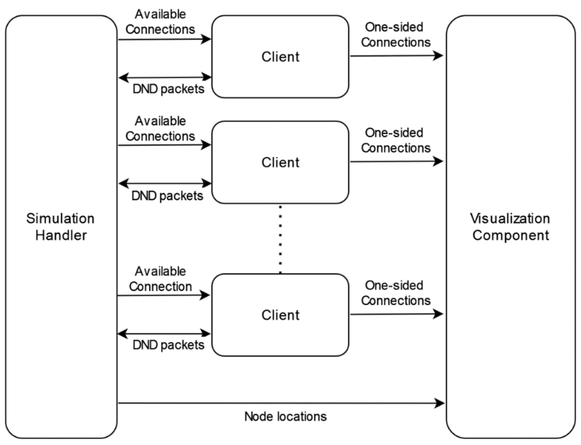


Figure 2: Initial structure of the DSM

Figure 2 represents the high-level structure of the DSM system. In this architecture, the Visualization Component handles communication with the Clients directly to receive the state of the connections between the nodes. To be able to develop additional components which would interact with the DSM without changing the high-level structure, would mean each additional module interfacing with the DSM would also need to handle these communications with the clients and sim handler separately, as well as changing the client and sim handler to be aware of the additional components. Additionally, the data required by multiple components would need to be duplicated across the system, instead of being stored and processed in a single component.

5.3. Architecture Style

For our project's purposes, we decided to use Data Centred Architecture. Data Centred Architecture is characterized by data being centralized and accessed frequently by other components, which possibly modify data. The primary goal of this style is to ensure integrity of data, meaning that all the data that is flowing through the system can be accessed only from a single source, preventing discrepancies between multiple components containing different data. A Data Centred Architecture consists of different components that communicate through a shared data repository. The components access shared data structures, meaning they are not dependent on each other to properly function. This architecture aims to organize and manage data in a way that ensures that it is accessible by other components and consistent across the whole system, while also keeping component coupling minimal.

The main advantages of the Data Centred Architecture are the following:

- **Adaptability and Flexibility**: Using a Data Centred Architecture ensures that components within the DSM are independent of each other's functionality and implementation, which allows for easy modifications and extensions.
- **Scalability**: This design allows us to easily scale the system to multiple devices, as we have a centralized state which can be accessed by multiple devices concurrently.
- **Data Consistency**: By using this architecture style, we can ensure that the data that flows within the components of the DSM is the same across the complete system. This means the data space will collect data from some components and then update the other components to ensure that the data's state is the same in all components of the DSM.

In the project's context, it allows us to reduce the number of connections between various components and simplify the development process for the current project as well as for future work. As a result of applying this design pattern, the components that are part of the DSM and require continuous updates need to be connected to a shared data space, in our case represented by the State Handler component. Moreover, we call our data space persistent since it will always store the previous state of the data, so we can compare it to the current state to determine whether the simulation changed and decide if we need to update the other components. This data space will be deployed in our system as a component that will continually update its local data structures depending on the incoming messages coming from the simulation.

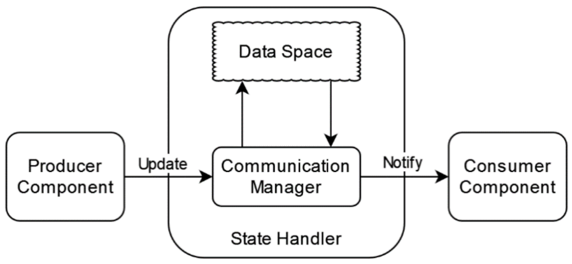


Figure 3: Producer / consumer model

In Figure 3, we show a simplified diagram of how we will incorporate this architectural style into our system. As shown, we will have two types of components that will communicate directly with the data space. The first type of component communication will only receive new data from the data space and update the component's local data, as this is shown in the diagram through the one-way arrow that points from the data space to the component. The other type will send generated data to the data space. For example, the Output Visualization Component will only retrieve the data from the data space, while Client Components will update the state of the connection data in the data space.

5.4. Final Design

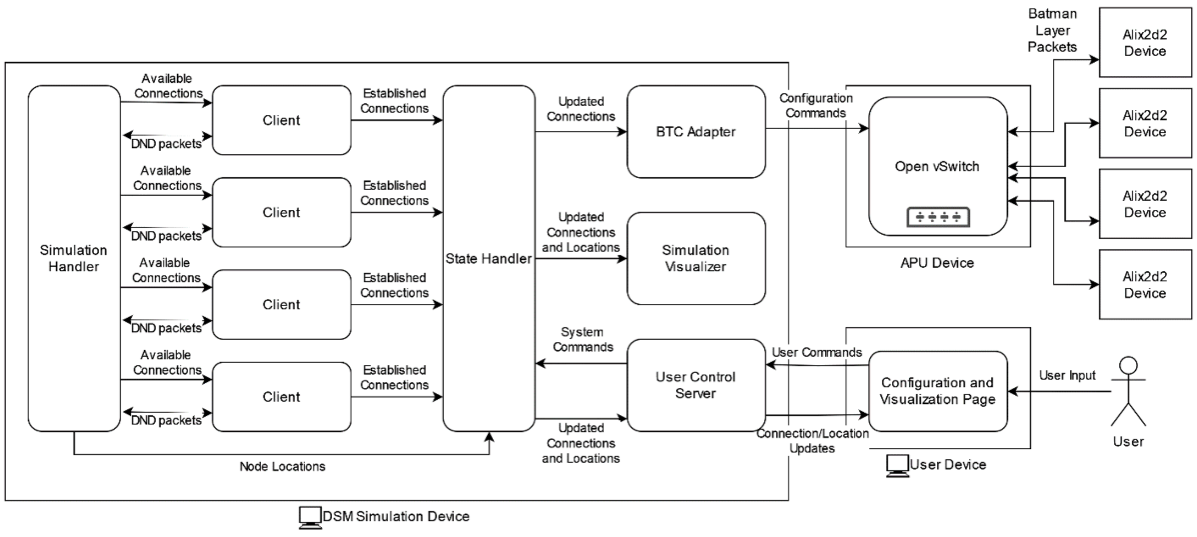


Figure 4: High level architecture of the system

In Figure 4, we present a high-level overview of the architecture we will use to implement the system. The main objective of this architecture is to have a State Handler that will store the connections between nodes and their locations. As a result, we can use the state handler to send the required information needed to change the topology of BTC and show the location of nodes in the graphical output components. Because of this decision, the clients are not required to interact directly with other components besides the Sim Handler and the State Handler. Therefore, this architecture still retains the main structure of the initial architecture, meaning that the initial DSM simulation code will require minimal changes. This allows for easy addition and connecting of new components, which will ease future development, while also reducing needed refactoring.

While currently in the design, the DSM simulation components live on the same device, it is important to note that because inter component communication is done through sockets, the system can be distributed on multiple devices if required, for example by running the BTC adapter on the APU device.

5.5. Components

As seen in Figure 4, the system is composed of the following components: Simulation Handler (or Sim Handler), Clients, State Handler, BTC Adapter, Simulation Visualizer, User Server / Configuration and the Open VSwitch. In this section, we will briefly articulate the primary functionalities and purpose of each component within the system.

- **The Sim Handler**
  It is mainly responsible for calculating the radio paths, using parameters like antenna gains and height, which are then sent to the Clients. Another essential responsibility is the calculation of the movement for each node. This includes taking the current locations of the node and recalculating the new locations based on the values of the velocity, rotation, and others. After the recalculation is completed, the new locations are sent to the state handler, which is done once for each iteration of the DSM.

- **The Client**
  Is responsible for receiving the peer valid paths and radio paths between other clients, which are generated inside the Sim Handler, and then calculate the valid links according to vacant frequencies and other factors. The latter is done by the DND module running an algorithm that determines which sector must start, stop or handover connections. For each iteration, the Clients send to the State Handler which other clients it is connected to and through which sector of the node and communication frequency the connection is established.
- **The State Handler**
  It is responsible for receiving connection data from all the Clients and position data from the Sim Handler. It processes the incoming data and then sends it to the components which require it. From the Sim Handler, it receives the newly calculated locations of the nodes, which are then stored and sent to the Graphical Output and User Control Server. From each client, it receives its current connections, which are stored locally. The state handler then verifies which pairs of clients are connected both ways and compares them to the previous connections. If a change is noticed, then the State Handler will send the required updates to the subsequent components. The state handler is also responsible for calculating the link bandwidth between each pair of connected clients and forward this value to the BTC adapter.
- **The BTC Adapter**
  This component is responsible only for receiving the added and removed connections and the connection bandwidth from the State Handler. It then is responsible for configuring the Open vSwitch based on this data. When the BTC receives the connections, it generates the commands needed for modifying the topology / link proprieties. Using secure shell (SSH) the commands are sent to the APU device, which based on this, then modifies the topology/proprieties of the network.
- **The Simulation Visualizer**
  This component represents the initial interface used by the DSM. It is responsible for displaying the nodes, active and inactive connections on the map while also showing the handover algorithm between the sectors of the nodes. Active connections are the ones that are currently used by the Clients to communicate, while the inactive ones are connections that are not used due to a lower signal strength.
- **The User Control Server**
  This component serves the newly developed web-based user interface. It is firstly responsible for serving the webpage used to configure and visualize the DSM system. It also handles user commands, such as starting / stopping the simulation. As such, this component is responsible for initializing and configuring the DSM system at each run.
  The webpage hosted by this server is mainly used as a configurator, where the user can interact with the map by adding nodes, waypoints, and movement metrics for each node. This information is saved in the form of a configuration file represented through JSON. The configuration's structure was customized by us to fit our needs and the context of the requirements. This page also allows visualizing the simulation, being an improvement over the initial Simulation Visualizer component.
- **The Open vSwitch**
  This component represents the virtual switch used for modifying the topology of the network running on BATMAN. This component runs the commands originating at the BTC Handler to change the topology according to the DSM simulation.

5.6. Inter-component communication

This section describes each interface of the architecture, which includes how each component is communicating with other components and how the data is moving within the system. Also, communication between every component of the system is done using sockets.

	Component 1	Component 2	Data Exchanged
1	Sim Handler	Client	The Sim Handler sends to the Client the newly calculated radio paths of the nodes.

2	Sim Handler	State Handler	The Sim Handler sends to the State Handler a JSON structure containing the type of the message and the connections of the client sending the message.
			{
			“type” — message type,
			“O” – locations,
			}

3	Client	State Handler	The Client sends to the State Handler a JSON structure containing the type of the message and the newly calculated locations of the nodes.
			{
			“type” - message type,
			“node_id” – id of the node that sent the message,
			“active” – the nodes to which the current node is connected,
			“inactive” – inactive connections of the node,
			“freqs” – the frequency of each sector of the node
			}

4	State Handler	BTC Adapter	The State Handler sends to the BTC Adapter a JSON structure containing the type of the message, and the added and removed connections inside the DSM.
			{
			“type” - message type,
			“new” – newly added connections,
			“removed” – removed connections,
			“updated” – bandwidth for each connection
			}

5	State Handler	Simulation Visualizer	The State Handler sends a JSON structure containing the type of the message, the added and removed connections inside the DSM, and the new locations of the nodes.
			{
			“type” - message type,
			“O” – locations,
			“active” – the nodes to which the current node is connected,
			“inactive” – inactive connections of the node,
			“freqs” – the frequency of each sector of the node
			}

6	User Control Server	State Handler	User Control Server sends to the State Handler commands regarding the DSM, for example starting or stopping the simulation. It also receives from the State Handler a JSON structure:
			{
			“type” - message type,
			“node_id” – id of the node that sent the message,
			“active” – the nodes to which the current node is connected,
			“inactive” – inactive connections of the node,
			“freqs” – the frequency of each sector of the node
			}

7	User Control Server	Configuration and Visualization	The Configuration Page sends to the Server a configuration JSON structure containing information about the locations of the nodes, their waypoints, and movement metrics. This file is read by multiple components at the start of the DSM simulation. It also can send commands to start or stop the simulation inside the DSM. The User Control Server sends to the Visualization Page a JSON Structure:
			{
			“locations” - message type,
			“connections” – active connections,
			“inactive” – inactive connections,
			“freqs” – frequency for each sector of a node
			}

8	BTC Adapter	Open vSwitch	The BTC Adapter sends SSH open flow commands used to modify the topology inside the BATMAN network. SSH traffic-control commands used to modify the link proprieties between nodes.

- **Interface 1: Sim Handler and Client**
  These components are communicating with each other to find new connections between the other Clients, based on metrics, like signal strength. Once the Sim Handler is connected to all the Clients, it starts running based on an iteration system, which includes handling incoming messages from Clients, mainly valid paths that were determined by the Clients in the previous iteration. Besides this, the Sim Handler also sends relevant radio paths and discovered peer sectors to other Clients.
- **Interface 2: Sim Handler and State Handler**
  Once the locations of the nodes have been updated, the Sim Handler will send a message to the State Handler that contains the new locations of the nodes. When the State Handler receives the message, based on the type of message, it will store the new locations in a local data structure.
- **Interface 3: Clients and State Handler**
  For each iteration, all Clients will send their current connections and through which sector the connection is established to the State Handler. When the State Handler receives the message through the socket, it will determine the two-sided connections and store them in a local data structure.

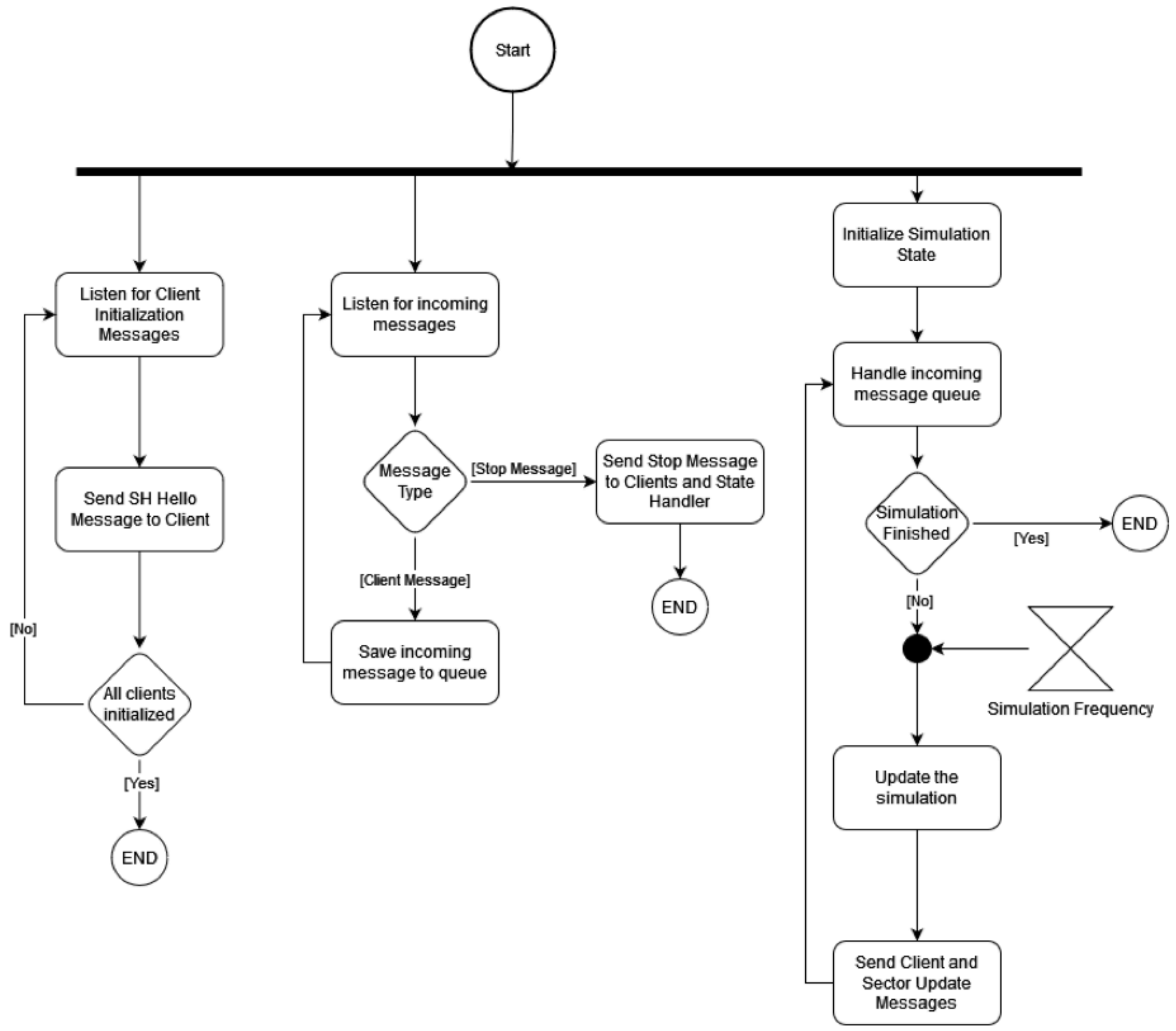
- **Interface 4: State Handler and BTC Adapter**
  After each iteration of the DSM, when the State Handler receives a connection data structure from a node, it will check if new connections have been added. In the case of new connections, the State Handler will send the new and removed connections between the clients to the BTC Handler, which will receive and then store them for further processing needed when sending commands to the OvS.
- **Interface 5: State Handler and Simulation Visualizer**
  With each iteration of the DSM, after new locations of the nodes are calculated and new connections are received from the Clients, the State Handler receives them and then sends the new locations, the current active and inactive connections to the Simulation Visualizer. Inside the Simulation Visualizer, the new locations, and connections will be processed, and then the representation will be updated according to the changes.
- **Interface 6: User Control Server and State Handler**
  Whenever the user decides to start or stop the simulation inside the DSM using a button on the webpage, the User Control Server will send to the State Handler a command to do the intended action. Also, with each iteration of the DSM, the State Handler will send to the User Control Server the updated locations, the frequency of each sector of every node, active and inactive connections that will then be saved locally using a data structure.
- **Interface 7: User Control Server and Configuration and Visualization Page**
  The User Control Server serves the Visualization Page by sending the locations and connections when the Page send a GET request with the path '/get_simulation'. Also, the Configuration Page can send commands to the User Control Server to start or stop the simulation inside the DSM by the sending requests with the following paths: '/start_simulation' (POST request) and '/stop_simulation' (GET request). The Configuration Page sends a POST request since it needs to send to the DSM a JSON configuration that stores information about nodes, their waypoints, and movement metrics for each node. At the start of the DSM, each component within the DSM will read the configuration file for specific data.
- **Interface 8: BTC Adapter and Open vSwitch**
  Whenever the State Handler receives new connections from a client, it will send the added and removed connections to the BTC Handler. Upon receiving both lists, the BTC Handler will generate SSH commands to be sent to the APU to modify the topology of the network running on BATMAN.

5.7. Further Decomposition

In this paragraph, we will discuss the more complex components by further decomposing their actions in activity diagrams to represent the control flow and actions each component takes during the system's runtime. They show what each component is doing at each point of its lifetime, and the decisions that affect its control flow. The choice to use activity diagrams was motivated by the nature of the components of the system, each having an event-based and concurrent structure. We believe this underlines the important implementation details and presents them in a easily understandable diagram.

- **Sim Handler**

The Sim Handler is mainly responsible for initializing the simulation, handling incoming client messages, calculating the radio paths between the clients and updating node locations. Most of the Sim Handler implementation / structure is left unchanged from the previous development done on the DSM. Because this component is responsible for a large part of running the DSM simulation, we have refactored the Sim Handler to decouple its functionalities from each other, by separating the calculation of node movement from the calculation of radio paths. This mainly allowed us to modify the movement of the nodes to allow waypoint-based movement independently of the radio path calculation. Secondly, it allows for the development of more complex features such as fast forwarding through the movement to a time point of interest, by either increasing the frequency at which the update function is called or implementing a different movement function.



Start
Listen for Client Initialization Messages
Send SH Hello Message to Client
All clients initialized
[No]
[Yes]
END
Listen for incoming messages
Message Type
[Stop Message]
Send Stop Message to Clients and State Handler
END
[Client Message]
Save incoming message to queue
Initialize Simulation State
Handle incoming message queue
Simulation Finished
[Yes]
END
[No]
Simulation Frequency
Update the simulation
Send Client and Sector Update Messages

Figure 5: Activity Diagram of the Sim Handler

▸ State Handler

Within the DSM, the State Handler represents the data space of the system, where we store all the current information about the connections and location of the nodes. The State Handler has a local dictionary that will store the required data and another one that will store the previous state of system, since we want to be able to find out which changes have been made to the system, so we can update the topology of the BATMAN and update the graphical output of the system. Initially the component will read the configuration file, in order to know the number of nodes and the number of sectors of each node. After that, the program will continuously wait for incoming messages though its socket. Once a message is received, it will read the type of the message and then act accordingly. In the case the type specifies that the data contains the location of the nodes, the local data structure will be updated with the new locations. If the type specifies that the message contains connection data, the local structure that keeps track of connections will be updated and compared to the previous state to find out the differences. After this, the differences will be sent to other components that need to modify the network topology or its representation. Additionally, for each connection it will compute the respective bandwidth, and in the case of a change will send that to the BTC handler additionally to the changed connections.

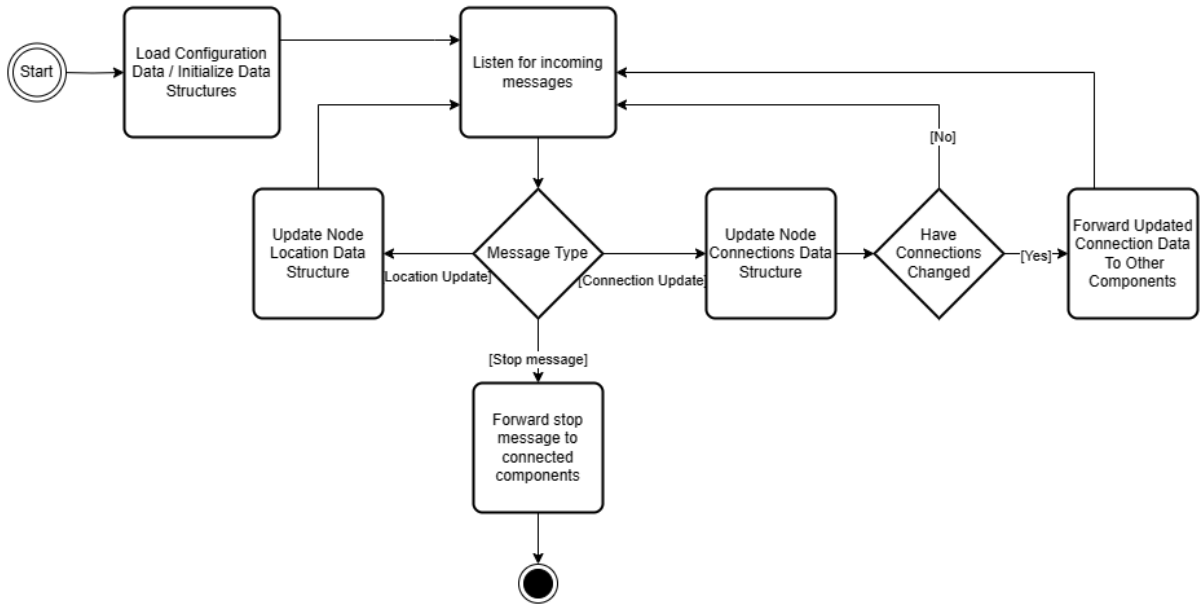

Start
Load Configuration Data / Initialize Data Structures
Listen for incoming messages
Update Node Location Data Structure
Message Type
[Location Update]
[Connection Update]
Update Node Connections Data Structure
Have Connections Changed
[No]
[Yes]
Forward Updated Connection Data To Other Components
[Stop message]
Forward stop message to connected components

Figure 6: Activity Diagram of State Handler

- **BTC Adapter**

As we have mentioned previously, this module is responsible for receiving the new and removed connections from the State Handler, and then sending commands to the Open vSwitch. When this component starts running, it will read the configuration file for the number of nodes that are present in the network. Following this, it will then continue waiting for incoming messages from the State Handler through its socket. When it receives a message, it will read its type and will act according. If the type of the message specifies that the message contains data about current connections, the function will generate multiple OvS commands for new or removed connections and Traffic Control commands for updated bandwidths and send them to the APU to change the topology inside the BATMAN network.

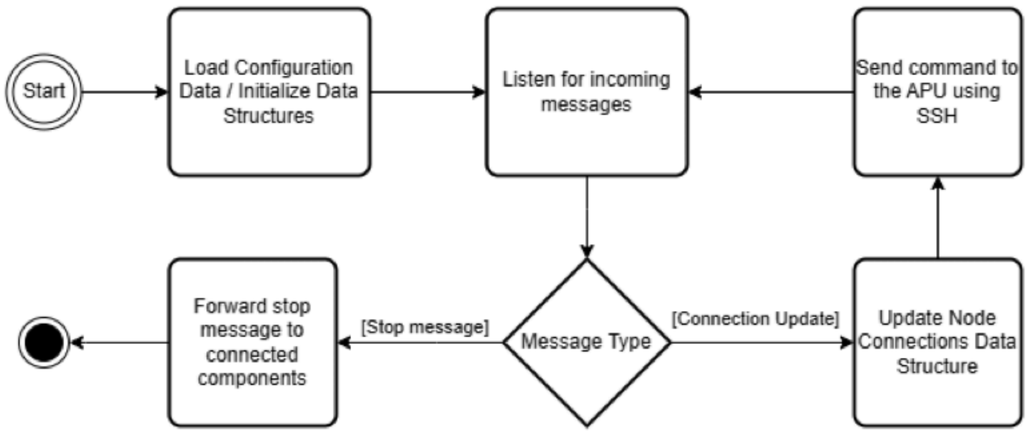

Start
Load Configuration Data / Initialize Data Structures
Listen for incoming messages
Send command to the APU using SSH
Forward stop message to connected components
[Stop message]
Message Type
[Connection Update]
Update Node Connections Data Structure

Figure 7: Activity Diagram of BTC Adapter

5.8. Technologies/Frameworks

- **Programming Language**
  To develop the integration between the systems, we will be using multiple technologies and frameworks. Newly developed components which rely on the existing DSM will be developed in Python, since the DSM is already implemented using this programming language. The decision to use Python for the development of the DSM was made by the team who previously worked on developing the system, as it allowed for easier implementation of the simulation visualization. As a result, they have re-implemented the system in Python, which was initially developed in C. Despite this, the drawing process was done using 'Matplotlib' — a plotting library for the Python programming language. As this library is primarily intended for plotting, it is highly inefficient when used for displaying dynamic systems which require multiple updates per second. Matplotlib imposes a performance penalty, being able to only display up to 20 nodes before it becomes unusable. Because of this, we have decided not to implement the new User Interface in Matplotlib, meaning that in the future the DSM system can be transitioned back into the C programming language, as it is more suitable for embedded systems and handles memory more efficiently. Unfortunately, this is out of the project's scope as time is a limited resource, so it is not possible to switch back to C, meaning that we will continue using Python for the purposes of this project.

- **User Interface**
  We have decided to implement the new User Interface using HTML and JavaScript. The decision was made since it would allow for a more efficient, user-friendly and responsive user interface. Besides this, keeping in mind future development, this would benefit the client a lot, as the interface could be easily ported to other devices, for example a phone or a tablet, and would allow accessing the simulation remotely.

- **Frameworks and Libraries**
  For mathematical computations that are done by the simulation method, we have chosen to continue using NumPy, as it is a universal standard used for manipulating numerical data in Python. To send shell commands through ssh more efficiently to the Open vSwitch using the BTC Handler, we have decided to use the Paramiko library. This library was created for common client use-cases like running remote shell commands.
  To simplify the development of the User Control Server, we decided to use the Bottle Python library, as it is very efficient, lightweight, and dependency-free. It allows us to easily specify API endpoints and statically host web pages.
  To easily display the output of the simulation, we decided to use d3.js — a JavaScript library which allows us to easily and dynamically manipulate scalable vector graphics.

- **Inter-process Communication** As we described above, our system consists of multiple components that are running in parallel, communicating and sending data between each other. The communication is done using sockets, which is a simple and an efficient enough way for our context to send data between programs and threads. The choice to use sockets for inter-process communication was done by the previous team working on the DSM, and motivated by the fact that this better aligns with how the system would work in reality. The messages that are sent are following a predefined protocol that specifies its type, so that the thread that receives the message knows how to handle the data. Also, the data that flows within the system is structured in JSON, because of its language-independent data format, as this would benefit when sending data to the Graphical Output used to make real-time changes to the User Interface.

- **Execution through iterations**
  Since the two of the main components of the system, the Sim Handler and the Client, are running based on a periodic execution, the other components are also updated according to this time-based

structure. At each iteration, both modules execute the same set of instructions in sequence, which includes collecting the data, processing it and then sending it to other components. Consequently, the Sim Handler receives data from them at each iteration, and in case of updated data, the other components will receive the new data. So, we can generalize and state that the whole system is running on a time-based scheme. Besides this, we have also separated the movement and calculation of radio path logic inside the Sim Handler. As a consequence, updating the locations of the nodes and calculating paths between the clients can be done at different iteration frequencies, this will not desynchronize both iterations since the radio paths will be calculated according to the last calculated location. As a result, this will help in the future with scalability and performance, but also with being able to easily modify the movement engine of the system.

5.9. Configuration and Visualization component Design Details

5.9.1. Introduction

The current DSM configuration and visualization component is limited and requires specifying the node's settings via a text input, which is not very intuitive. Also, current input only considers the node's movement in a linear fashion, and is statically defined initially, not allowing for changes in the movement based on time.

The new improved configuration and visualization component will have the following improvements:

- Platform proprieties
  - Nodes will be defined based on their platform properties
  - Platforms define the max speed and angular velocity of the node
- Graphical based input
  - Move nodes by means of drag and drop
  - Adding/removing nodes using buttons
- New route configurator
  - Configure the node's route by means of waypoints
  - Each waypoint's arrival time can be specified
  - Copying a route and pasting it to another node
- Loading and saving configuration

The node configuration window will contain text-based input with the following proprieties options per node:

- Number of sectors
- Initial node's coordinates and rotation
- Rotation speed
- Translation speed

5.9.2. Hi-Fi prototype

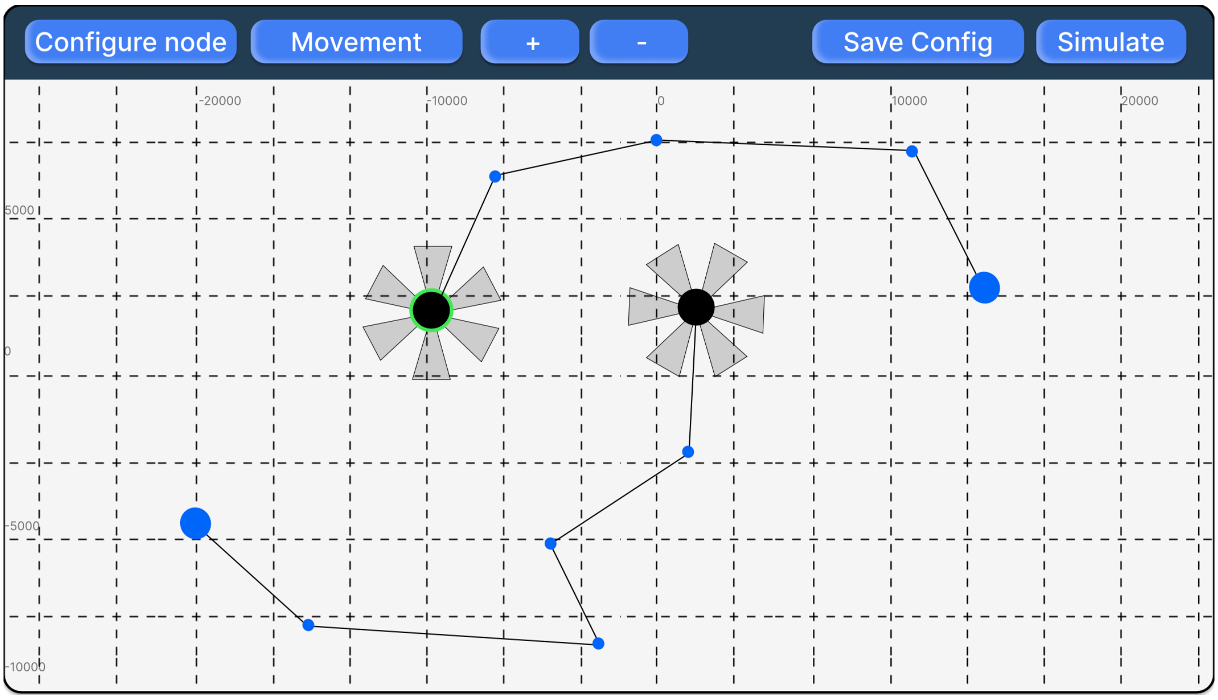
We designed a hi-Fi prototype to specify the general look of the new input generator. Currently, there are 2 options for the configuration and visualization component user interface, the main difference between them being the position and shape of the navigation bar menu.

The interaction with the interface will be as follows:

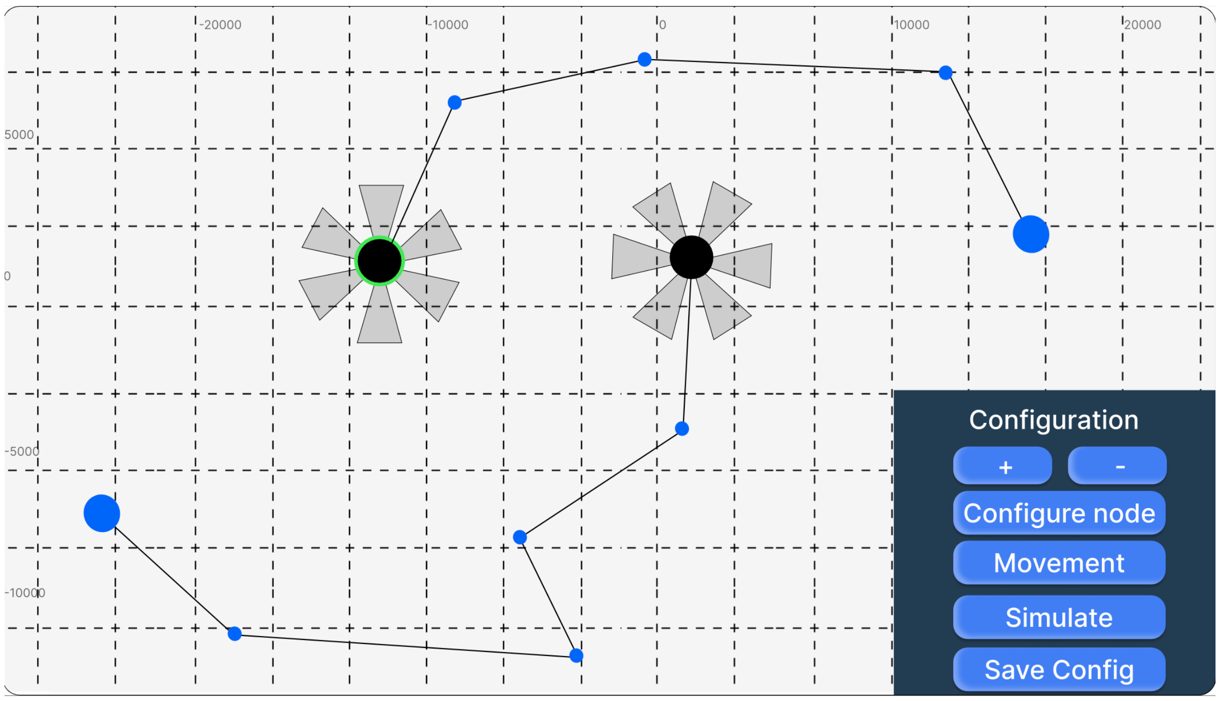
- The user can click on a node to select it
  - After the node is selected, user can delete it by clicking '-' sign
  - User can configure node's properties by clicking on 'Configure node' button
  - 'Movement' button allows the user to specify a route for the node, after clicking on it:
    - To create a waypoint, user can click on a canvas and a waypoint would be created

    · To adjust a node's location, the user would need to click on the waypoint and drag & drop it
    · To delete a waypoint, user would need to select the waypoint and click on the '-' button
    · To adjust arrival time at a waypoint, user would just click on a way point and enter time in the appeared window
- The user can add a new node with a '+' sign
- The user can save the configuration with 'Save Config' button
- To start the simulation, user would need to click on 'Simulate' button

5.9.2.1. Option 1

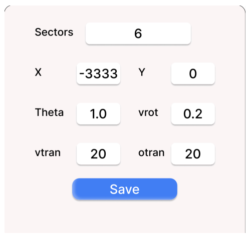


5.9.2.2. Option 2



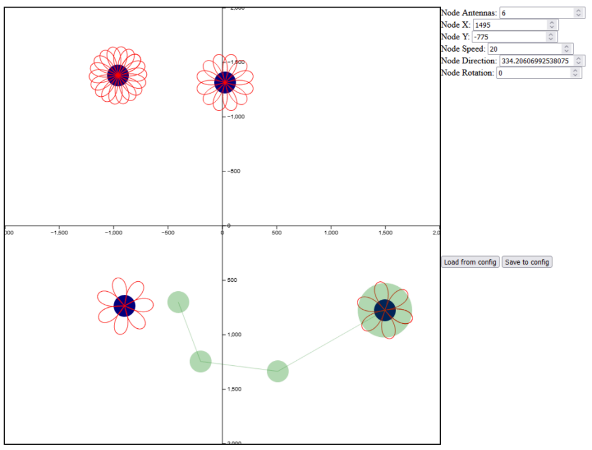
5.9.2.3. Window for configuring a node

The window for configuring a node allows the user to update its properties, such as the number of sectors on the node's antennas, initial location in terms of the X and Y coordinates, Theta (direction of a ship's bow), Vtran (translation speed) and Vrot (rotation speed). After the user is done with inputting the node's settings, they need to click save.



Sectors: 6
X: -3333 Y: 0
Theta: 1.0 vrot: 0.2
vtran: 20 otran: 20

5.9.3. Prototype



Node Antennas: 6
Node X: 1495
Node Y: -775
Node Speed: 20
Node Direction: 334.20606992538075
Node Rotation: 0

A prototype was developed to showcase the client usability of some new features specified previously, including adding new nodes and waypoints using mouse clicks, drag and dropping nodes and waypoints, and easily removing them. The prototype served a purpose of getting feedback from the client and also as an initial backbone of the actual implementation.
This prototype only displays how the user would interact with the graphical user interface; it does not highlight the final user interface design.

5.9.4. Configuration data format

The configuration and visualization will also define a JSON file that will be further used in the DSM to specify the program what the initial locations of the nodes are, their movement definitions, and the locations and the arrival time of the waypoints. When the DSM starts the simulation, it will read the configuration file and set up the movement for each node. We defined a new structure for the JSON configuration to fit with our new requirements and not interfere with the current Maritime Manet Simulation component. The old configuration file had a different structure which did not facilitate waypoint based movement.
Breakdown of its structure:

- `ships`: An array containing objects representing individual ships.
  - Each ship object contains the following properties:
    - `initialX`: The initial x-coordinate of the ship's position.
    - `initialY`: The initial y-coordinate of the ship's position.
    - `speed`: The speed of the ship.
    - `rotation`: The speed rotation of the ship.
    - `initialR`: The initial facing radius (orientation) of the ship.
    - `antennaN`: The number of antennas on the ship.
    - `ship`: An object containing the ship's current position and radius.
      - `X`: The current x-coordinate of the ship's position.
      - `Y`: The current y-coordinate of the ship's position.
      - `R`: The current facing radius (orientation) of the ship.
    - `waypoints`: An array containing objects representing the ship's waypoints.
      - Each waypoint object contains:
        - `wx`: The x-coordinate of the waypoint.
        - `wy`: The y-coordinate of the waypoint.
- `world_X`: The width of the canvas.

- `world_Y`: The height of the canvas.

5.9.5. Activity diagram

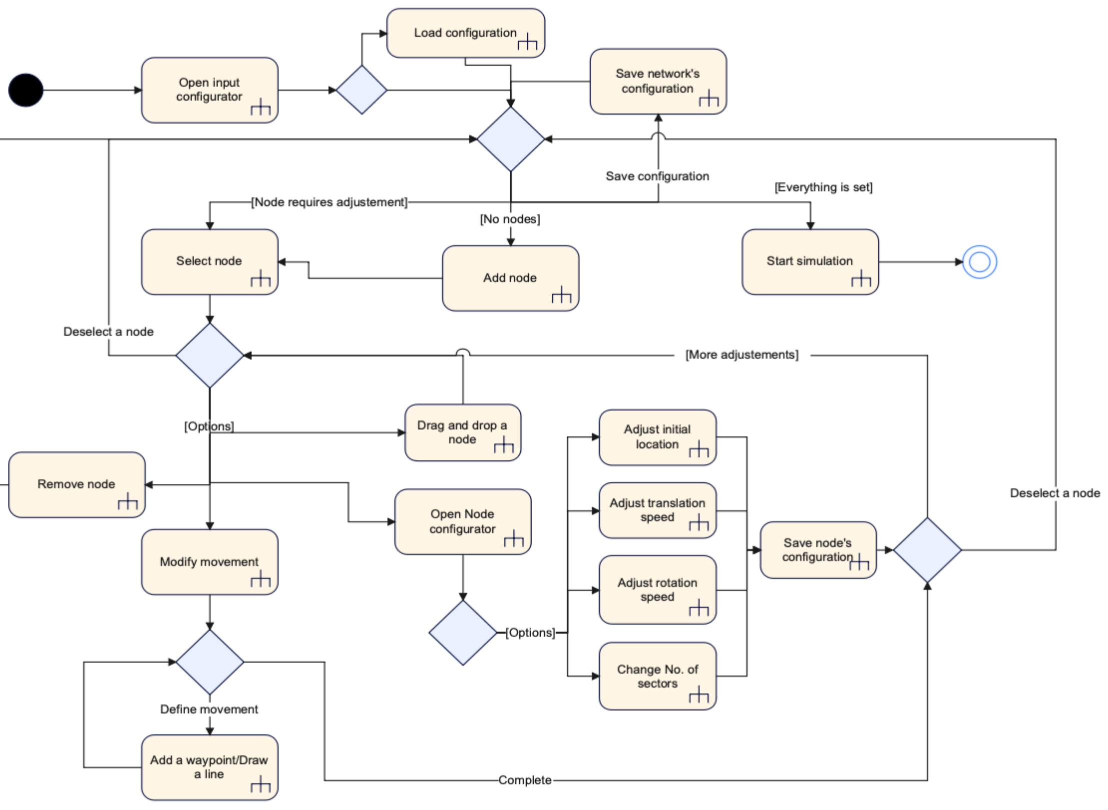


Figure 8: Activity Diagram for configuration and visualization

Description of Figure 8:

1. The user opens the input generator
2. He either loads a configuration or skips this step
3. Then he can either start the simulation (given that there are some nodes), or he can select a node or add a new node
4. If a user creates a new node, then that node is automatically highlighted as it is selected
5. A selected node can either:
   - Be removed
   - Change its route (route modification details are not captured in this diagram):
     - A waypoint can be created
     - Waypoint can be deleted
     - Waypoint can be moved
     - Waypoint arrival time can be modified
   - Be dragged & dropped
   - Be configured with node configurator window:
     - Initial location adjusted with X and Y coordinates
     - Translation speed can be modified
     - Rotation speed can be modified
     - Number of sectors can be specified
6. Then the user will go back to step 3

5.9.6. Use cases

	Use Case	Description
1	Start Simulation	If a user is done with configuring the network simulation, then they can click on 'Simulate' button and the simulation will start. During the simulation, the user cannot make changes to the nodes, they can only stop the simulation.
2	Select Node	User can select a node (for further manipulations) by clicking on it.
3	Create a node	User can create a new node by clicking on a button.
4	Delete a node	Selected node can be deleted by clicking on delete button.
5	Configure node position	Selected node's position can be configured either manually (with X and Y coordinates) or with drag & drop.
6	Configure position manually (with x and y coordinates)	Selected node's position can be manually entered in the configuration window utilizing X and Y coordinates on a plane.
7	Configure position graphically (drag and drop)	Selected node's position can be adjusted by dragging and dropping the node in the space.
8	Select number of antennas from a drop-down list	a Selected node's type of antenna can be changed by selecting an option from a drop-down list in the configuration window.
9	Specify translation speed	Selected node's translation speed can be specified in the configuration window.
10	Specify rotation speed	Selected node's rotation speed can be specified in the configuration window.
11	Movement configuration	Users can configure node's movement parameters for nodes within the simulation environment, such as translation speed and rotation speed.
12	Draw a path of a node with waypoints	Selected node's path can be specified by creating waypoints, to do that a user can just click on an empty space in the canvas
13	Confirm path	Confirm and finalize the drawn path for a node within the simulation environment, ensuring that it follows the intended trajectory during simulation.
14	Clear path	Selected node's path can be cleared where all waypoints are deleted.
15	Save configuration	Saves the current network configuration to a JSON file.
16	Load configuration	Loads a JSON configuration file and shows the network situation on the plot, allowing the user to adjust it or start the simulation.

Table 1: Configuration and visualization brief use case descriptions

5.9.7. Use case diagram

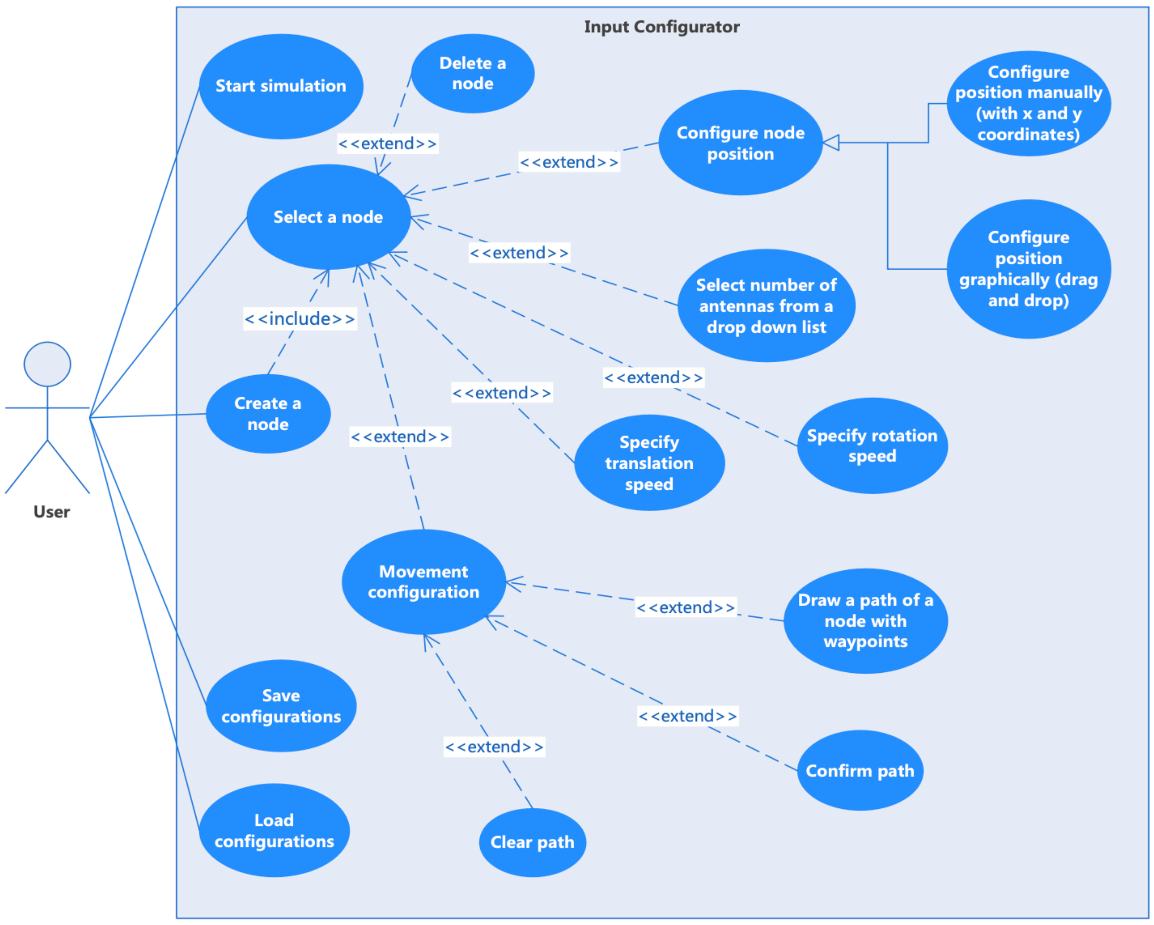


Figure 9: Use Case Diagram for configuration and visualization

Figure 9 generalizes all the use cases that would be available to the user with the configuration and visualization component. While we might later adjust small features, the overall usability of the configuration and visualization will remain the same.

<< **Include** >> tag means that the use case is mandatory, so if a node is created that means that a node must be automatically selected as well (after it is created)

<< **Extend** >> tag means that the use case is optional, so for example, entering a movement configuration mode does not necessarily mean that a waypoint must be created by default, a user can enter and leave the movement configuration mode without adjusting the route whatsoever.

5.9.8. Design choices

The design choices for the new configuration and visualization component were made to address several limitations of the current system and to enhance user experience.

- **Web-based user interface**: the decision to develop a web-based user interface comes from its inherent scalability and adaptability, offering potential expansion into a mobile application in the future. Using programming languages like JavaScript and HTML provides significant flexibility and customization options. Additionally, the team's familiarity and experience with these technologies simplifies the development process. Although alternatives such as Pygame or Vispy were considered, the widespread popularity, modularity, and ease of JavaScript and HTML made them the preferred choice to other options.

- **Waypoint based movement**: the decision to implement waypoint-based movement, rather than drawing paths with a mouse or using command-based controls (e.g., right/left/up/down), was made after consulting with the client to determine the most suitable method for end-users. Through discussions with potential end-users, it was established that waypoint-based movement aligns best with the domain of the application, as it is widely used and easily comprehensible. Despite its limitations, such as difficulties with sharp corners and limited curvature, waypoint-based movement offers convenience and ease of use compared to drawing paths with a mouse or using command-based controls, which may lack precision or be time-consuming.
- **Available node options**: the majority of node options, such as number of sector, translation, and rotation speed were retained from the existing DSM input configurator.
- **JSON Configuration Format**: the new JSON configuration format ensures compatibility with the existing DSM.
- **Graphical user interface**: the main design choice was to make the new input configurator graphically based, so that the user's can more intuitively specify the network situation and interact with it visually. Functions like dragging and dropping nodes, adding/removing nodes, and configuring routes via waypoints enhance usability and intuitiveness.

6. Results

6.1. DSM & BTC integration results

The implementation of the DSM & BTC integration stayed true to the design. To recap, the features it implements are:

- Connections between the ALIXes are dynamically created/updated/deleted according to the outputs of the DSM simulation.
- The connections also dynamically set their bandwidth according to what would be the realistic bandwidth based on the wireless communication specification.

Implementation details:

The details of how this ended up being implemented are as follows. We opted for doing the computation of which ALIX is to be connected to which ALIX on the laptop also running the simulation. This was done mostly due to ease of development. This way we would not have to update anything on a different machine, keeping the development environment limited to just our own laptops.

The tooling used to make/update the connections on the BTC were ovs-ofctl (openflow) and tc (traffic control). The openflow tool was chosen by the previous student who worked on the BTC. It allows us to change the connection between nodes using just the command line, which would prove useful since we could then use SSH to update the APU remotely. Secondly, we use the tool traffic control. This comes with a module called NETEM, short for network emulation. Through this, we can configure certain interfaces to adopt network conditions. We dynamically use this to set different bandwidth conditions, simulating congestion/bad connection at sea. As well as setting a default rate of packet loss of 2%. Again, this tool is also command line based, allowing us to use SSH for both of these.

Lastly, since the computation of which ports should be connected to which other ports is done on the user device and not the APU itself, we needed a way for assigning and keeping track of what is happening on the APU. For the first iteration of this system, we dedicated a port per ALIX to each other ALIX, which we were able to virtually connect when required. This kept a lot of the implementation simple but wouldn't be very scalable, so we updated this to a new system. In this system, each ALIX has x ports, where x is the max amount of other nodes it can be simultaneously connected to. Determined either by how many other nodes or how many antennas, whichever is smaller. And the system would then randomly assign unassigned ports for connections, keeping the required number of ports per ALIX down, even if the system were to scale massively.

6.2. Configuration and Visualization interface results

6.2.1. User interface

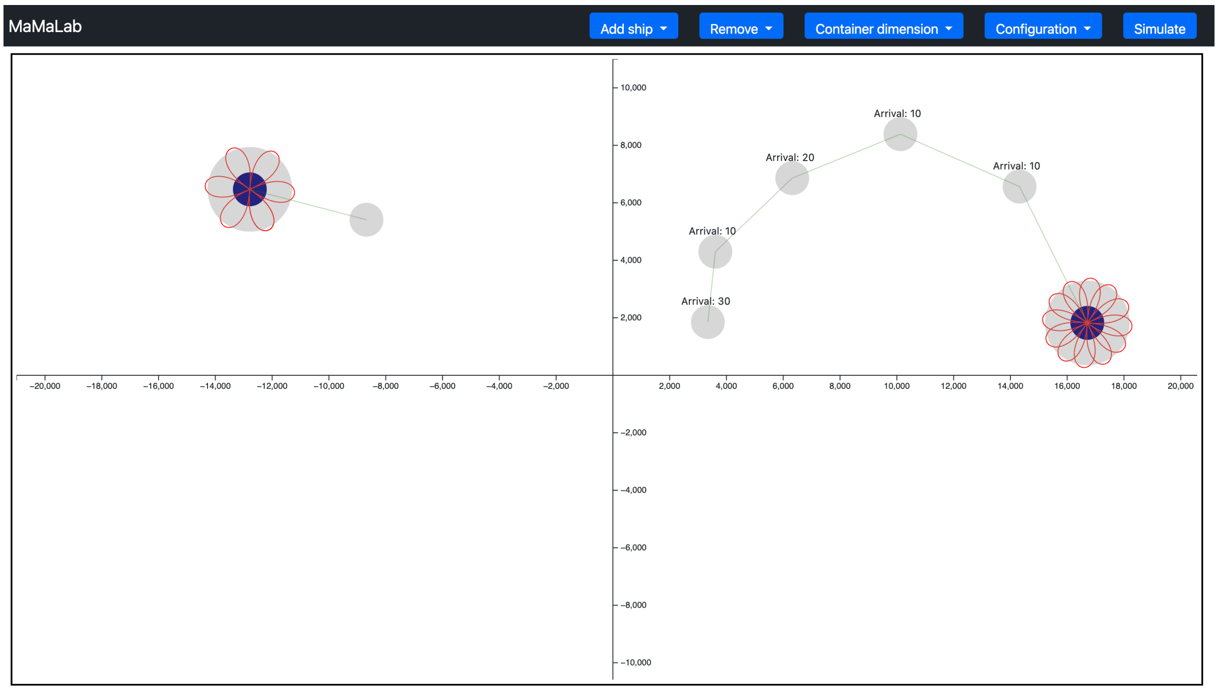


Figure 10: Final user interface

6.2.2. Design comparisons and added features

The final design varies a bit from the hi-fi prototype, as can be seen in Figure 10. Among the added features we have:

- Add node is replaced with add ship. The ship's size defines the number of antennas; the user cannot change this value.
- There are more removal options under the remove button such as: selected ship, last waypoint of selected ship, all waypoints for selected ship or all ships and waypoints.
- The configuration window was removed as per client's request, since there's no more need for that.
  - Arrival time can be added to each waypoint. The speed of the node is dictated by the distance and arrival time at the waypoint. If arrival time is not specified, the ship uses its maximum speed.
- The scale of the graph can be adjusted by clicking on the 'Container Dimension' button.
- Hotkeys are added for user's convenience, for example, when a node is selected and a backspace or delete key is pressed then the selected ship is removed.
- To deselect a node and finalize the route, user has to on 'Finish Route' button.

6.2.3. Implementation details

We have utilized HTML and JavaScript for setting up network's environment within a container. It utilizes the D3.js library for SVG manipulation, such as drawing nodes, waypoints and moving the nodes. The JavaScript code contains various functionalities, including coordinate transformation, container setup, ship management (such as adding, dragging, and removing ships), waypoint management (including adding and dragging waypoints), configuration management (loading and saving configurations), and event handling.

The HTML content includes a navbar with buttons for adding ships, removing ships/waypoints, setting container dimensions, and managing configurations, along with an SVG container for visualization and a DIV element for displaying a hint to add waypoints.

6.2.4. Visualization results

Finally, we have implemented the visualisation. Now, the new visualisation was done in the same page instead of presenting results in a Matplotlib video.

Implementation details:

The simulation and visualization of network behaviour were done through communication between client-side JavaScript and server-side Python code. The Python backend, used the Bottle framework, received requests from the frontend to start and stop simulations, and provided simulation data upon request. It also managed the state of the simulation and handled communication with external systems. On the frontend, JavaScript code visualised the simulation process, sending requests to start and stop simulations, and periodically fetching and processing simulation data from the server. This data was then used to update the visualization of ships' positions, orientations, and connections on the user interface.

7. Testing & Validation

7.1. Testing strategy

Because of the varied domains tackled by our project, we will consider a different testing strategy per component.

7.1.1. State handler

Objective: validate that incoming messages are handled and processed correctly, and are sent to the appropriate consumer components in the expected format.
The state handler is responsible for managing the communication between components, as well as aggregating the incoming data and processing it before sending it to the consumer components. As such, we decided to use black-box testing to verify its behaviour based on incoming packets by monitoring the output in its sockets. To achieve this, we used the `unittest` python module, to mock input and output without the need to run the whole system.

7.1.2. BTC handler

Objective: verify the accuracy of the data used for generating sent commands.
The BTC handler interacts with an external system, which poses difficulty to fully tests its behaviour. As such, we decided to use unit testing for its internal processing, and verify the data it uses to generate the sent commands.

7.1.3. BTC network

Objective: verify the correctness of the batman network environment generated using the DSM simulation.
Testing the network between the Alix devices was done with the `batctl` command line utility, which allows to list the direct neighbours of each device and to ping trace between devices to verify that the correct topology is used by the actual network. To test that the correct bandwidth is used for each link, we used the `iperf` command line utility, which given a `server` and `client` device, monitors the max bandwidth and other link proprieties between them.

7.1.4. Configuration and Visualization

Objective: validate configuration and visualization functionality, usability, compatibility, and performance across the entire system.
To test the configuration and visualization component, we will utilize following testing method, to verify that the configuration and visualization component is working as expected:
End-to-end testing: testing the entire application from start to finish, simulating real-world user scenarios. It verifies the functionality, performance, and reliability of the system as a whole.

	Testing	Description
1	Functional testing	▸ Verify that all features and functionalities work as expected according to the provided use cases. ▸ Ensure that user interactions produce the intended outcomes without errors.
2	Compatibility testing	▸ Test the component across different browsers (Chrome, Firefox, Safari, etc.) to ensure compatibility. ▸ Verify that the component functions correctly on different operating systems (Windows, macOS, Linux).

3	Usability Testing	▸ Evaluate the user interface for intuitiveness and ease of use. ▸ Confirm that users can perform tasks efficiently without confusion.
4	Performance Testing	▸ Assess the component's performance under various conditions (e.g., different network sizes, node configurations). ▸ Measure the response time for user actions and ensure that the application remains responsive.

Table 2: Configuration and visualization testing strategy

Table 2 describes testing strategy for configuration and visualization. This type of testing aims to simulate real-world scenarios and user interactions to ensure that the component meet the testing objectives.

7.2. Testing plan

7.2.1. State handler

	Test case	Description
1	Handle Location Messages	Test if the handle_msg function in the state_handler module correctly processes a message of type LOCATIONS. This test simulates the scenario where a message containing location data is received. The test initializes a mock message object with type LOCATIONS and a set of coordinates. It then encodes this message into bytes and simulates the reception of this message. The test ensures that the message is sent to the appropriate addresses defined in the configuration. Additionally, it verifies that the locations attribute in the state_handler module is updated with the received coordinates.
2	Handle Connection Messages	Test if the handle_msg function correctly processes a message of type CONNECTIONS. This test simulates the scenario where a message containing connection data is received. The test constructs a mock message object with type CONNECTIONS, including active connections, node ID, frequencies, and sequence number. It then encodes this message into bytes and simulates its reception. The test ensures that the message is forwarded to the appropriate addresses defined in the configuration. It also validates that the connections and frequencies dictionaries in the state_handler module are updated correctly with the received data for the respective node ID.

Table 3: State handler testing plan

Table 3 outlines a systematic testing approach for the functionality of handling messages in the state handler module. Each test case corresponds to a specific message type scenario and verifies the correct processing and propagation of data within the system. The tests are designed to ensure comprehensive coverage of message handling functionalities, validating both the sending and updating of data attributes. Following execution, the tests contribute to ensuring the robustness and reliability of the system's communication mechanisms.

7.2.2. Configuration and Visualization

	Test case	Description
1	Start Simulation (Use Case 1)	Click on the “Simulate” button and verify that the simulation starts in the same page. Confirm that user cannot change any configurations, but only stop and pause the simulation.
2	Node Selection (Use Case 2)	Click on a node and verify that it becomes selected. Ensure that only one node can be selected at a time. Test deselecting a node by clicking ‘Finish Route’ button.
3	Node Creation (Use Case 3)	Click on the “Add ship” button, select from a drop-down list and verify that a new node is added to the canvas of correct type. Test creating multiple nodes and verify their placement.
4	Node Deletion (Use Case 4)	Select a node and click on the “Remove” button, select “Selected ship”. Verify that the selected node is removed from the canvas. Repeat the test, but instead using ‘del’/’backspace’ key on the keyboard.
5	Node Position Configuration (Use Cases 5-7)	Test adjusting node position graphically by dragging and dropping. Verify that the node’s position updates accordingly.
6	Antenna number (Use Case 8)	When creating a node ensure that the created ships have a correct number of antennas according to its size.
7	Movement Configuration (Use Cases 9-11)	Specify time of arrival for a selected node. Verify that the node’s movement parameters such as maximum speed are applied during simulation.
8	Path creation (Use Cases 12-14)	Draw a path for a selected node by creating waypoints. Confirm the drawn path and ensure it follows the intended trajectory. Test clearing the node’s path and verify that all waypoints are removed from the JSON file.
9	Configuration Saving and Loading (Use Cases 15-16)	Save the current network configuration to a JSON file. Load a JSON configuration file and verify that the network situation is restored in the browser.
10	Cross-Browser and Cross-Platform Testing	Test the component on different browsers and operating systems to ensure compatibility.
11	Usability Testing	Evaluate the user interface for usability and user-friendliness.
12	Performance and Load Testing	Assess the component’s performance in various scenarios, like large network sizes. Measure the response time for user interactions and ensure acceptable performance levels.

Table 4: configuration and visualization testing plan

Table 4 describes a step-by-step testing plan for each of the use cases to ensure that each use case is covered, and each testing scenario follows one of the testing strategies. The tests are carried out by the development team and usability is verified with the client to ensure that the interface is user-friendly and easy to understand.

7.3. Testing results

7.3.1. State handler

	Test case	Passed	Result
1	Handle Location Messages	✓	Unit test passed successfully.
2	Handle Connection Messages	✓	Unit test passed successfully.

Table 5: State handler testing results

Table 5 describes the testing results for the state handler. All tests passed successfully, thus the state_handler module correctly processes messages of type LOCATIONS and CONNECTIONS.

7.3.2. Configuration and Visualization

	Test case	Passed	Result
1	Start Simulation	✓	When a simulate button is clicked, the page is updated and user options are turned off. The simulation starts.
2	Node Selection	✓	The node gets highlighted in light green when selected. Only one node is selected at a time
3	Node Creation	✓	Node creation works. The created node corresponds to the selected type of the drop-down list.
4	Node Deletion	✓	Node deletion works. Only one node can be deleted at a time, unless delete all nodes and waypoints option is chosen.
5	Node Position Configuration	✓	Single node can be dragged and dropped. It's waypoints positions are adjusted as well.
6	Antenna number	✓	Each type of ship is associated with a number of antennas. Correct antenna numbers are on the canvas per each node type.
7	Movement Configuration	✓	Arrival time can be specified per each waypoint and the node's speed is bounded by the maximum.
8	Path creation	✓	Waypoints can be created sequentially, last waypoint can be removed using a delete button and associated option. Correct trajectory is followed in the output. When waypoints are removed, the configuration file is updated.
9	Configuration Saving and Loading	✓	Saving configuration prompts the user to save the file wherever they want, in a correct JSON format. Loaded configuration overwrites the canvas and allows users to adjust the network situation.
10	Cross-Browser and Cross-Platform Testing	✓	The configuration and visualization was tested on: Linux, macOS and Windows, in following browsers: Chrome and Firefox.
11	Usability Testing	✓	The interface is responsive and easy to understand. The buttons are self-explanatory. Can be improved by adding the manual into a dialogue window/pop-up for the user explaining all the hot-keys and features.

	Test case	Passed	Result
12	Performance and Load Testing	!	Generally the application is working as expected. But, has responsiveness issues, such as zooming in and out, can affect the canvas look. While refreshing solves the issues, it means that user's progress would be lost, unless the user saves the configuration beforehand.

Table 6: Configuration and visualization testing results

Table 6 describes the testing results. The majority of tests passed, and all essential features performed as intended, providing users with flexibility and control over network simulation. Despite minor responsiveness issues during zoom operations, the module generally delivers as expected, with refreshing acting as a solution to any unexpected glitches.

8. Future work

In this section we will discuss further work to be done on this project, opportunities we found during development, or suggestions for future developers.

8.1. DSM - BTC integration

The integration between the DSM and BTC relies on the DND algorithm to compute active connections between nodes, but also on the RSSI value of each link to compute the quality of the connections. As the DSM was not initially designed for the purpose of producing this data, the code surrounding this could be further improved to separate and properly document these two functionalities. The code for the computation of the link proprieties can be further improved to take into account parameters like loss and delay in a dynamic way based on the simulation.

8.2. BTC Virtualization

The current physical system of the BTC relies on using Alix2d2 embedded devices. This limits both the total number of nodes that can be simulated, and the processing that can be run on these nodes. Development of applications that rely on this network would need to keep the limitations of these devices into account. A proposed solution to this problem is either to entirely virtualize the Alix devices, meaning the whole system should be runnable on a single machine, or to use the Alix devices as proxies for virtual machines that would use the network for communication between each other.

8.3. Configuration page

In the current state, the configuration component is suitable to verify various scenarios for Maritime-Manet, but improvements to the user interface and movement simulation are advisable. The following are the main points to be addressed in the future:

- Overlapping elements in the input configurator sometimes has unexpected or buggy behaviour. This often happens when trying to switch between ships before finishing the route, or trying to create overlapping waypoints for ships.
- Certain scenarios would require more fine-grained or specific movement, such as nodes moving together, synchronizing or waiting in a certain spot before continuing. Currently, this has to be done manually by the user, but dedicated movements for such use cases would be advisable, as they make the job of the user much easier.
- The current movement was designed to be able to handle the required scenarios, i.e. splitting and joining of groups of ships, testing hand-over between ships with constantly changing network topology. Moving to a more realistic movement simulation would be beneficial for users with experience in the navy domain. This could be done by defining different movement logic depending on the platform of the node.
- Additionally, the platforms currently defined serve for testing purposes only, and do not represent the actual proprieties of a boat. Defining these proprieties based on real life data would be advisable for improving the accuracy of the system.

9. Reflection On Process

9.1. Tasks distribution

	B. Colţa	**D. Negru**	**M. Spinei**	**N. Jongejan**	**Y. Alyoshyna**
DSM integration	✓		✓		
BTC integration				✓	
Architecture design	✓		✓		
Configuration design		✓			✓
Web user interface		✓	✓		
Testing	✓		✓	✓	✓
Report	✓	✓	✓	✓	✓
Poster	✓	✓	✓	✓	✓
Presentation slides	✓	✓	✓	✓	✓

9.2. Group reflection

As a team, we faced a steep learning curve within a limited timeframe, picking up various new concepts and technologies in the early stages of the project. This initial learning process resulted in some tension as we worked to figure out unfamiliar concepts. However, as the project progressed, we successfully developed many features, with significant progress made in the second half of the project.

We are extremely grateful for the incredible support and guidance provided by our client, who has been actively involved since day one, offering insights and helping us refine our work. His involvement played a significant role in shaping the project's direction and outcomes. Individually, we have learned a lot about real-world's companies approach to requirements elicitation, writing project proposals and designing system architectures.

Overall, we are pleased with the results we achieved as a team. Despite the challenges we faced, each of us made contributions using their unique skill set, thus together we ensured the project's completion. Moving forward, we recognize the importance of better balancing the workload distribution and continue to appreciate the collaborative effort.

10. Conclusion

In conclusion, the design report demonstrates the successful completion of the project, adhering to the waterfall project management approach (with some minor changes). The integration of the Distributed Simulator for MaritimeManet (DSM) and the Batman Topology Configurator (BTC) stayed true to the project's scope and objectives, achieving seamless communication and dynamic topology adjustments between ALIXes.

The integration process followed a structured approach, progressing through phases of requirements analysis, system design, implementation, testing and documentation. The deployment and evaluation phases fall outside the scope of our project. Weekly client meetings ensured alignment with the client's expectations and minimized risks associated with changing requirements and technical challenges.

The user interface underwent many changes, offering intuitive configuration and visualization features. It provides a user-friendly environment for setting up and visualizing network configurations in real-time. The interaction between the web client-side and the server-side allows for smoother dynamic visualization of network behaviour in comparison to the initial Matplotlib visualization.

In summary, the integration of the DSM and the BTC, paired with the new configuration and visualization interface, provides Thales with a true digital twin for testing and simulating swarm applications in a lab environment.