HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

School of Information and communications technology

Project Report

Net Formation In WSN
Graduation Research II

Vũ Đức An - 20215174

Hanoi, February, 2025

Table of contents

1 Introduction

The rapid advancement of the Internet of Things (IoT) has paved the way for innovative applications in various fields, including smart agriculture. One of the key challenges in modern farming is the efficient monitoring of environmental conditions such as temperature, humidity, soil moisture, and light intensity. Wireless Sensor Networks (WSNs) provide a scalable and cost-effective solution by deploying a network of sensors across agricultural fields to collect real-time data. This data enables farmers to optimize irrigation, detect anomalies, and improve crop yields.

1.1 Motivation

This project aims to design a new communication protocol for WSNs in an agricultural IoT application. Existing protocols often face challenges related to energy consumption, data reliability, and network scalability. Our proposed protocol seeks to address these issues by optimizing data routing and transmission efficiency while ensuring reliable communication among sensor nodes.

1.2 Objective

The following sections will present the background and related work, system architecture, details of the proposed protocol, implementation, results, and analysis. Through this research, we aim to contribute to the development of more efficient and sustainable IoT-based smart farming solutions.

2 Problem

Wireless Sensor Networks (WSNs) have become an essential technology in precision agriculture, enabling real-time monitoring of environmental conditions to optimize farming operations. By deploying a network of sensors across agricultural fields, farmers can track critical parameters such as soil moisture, temperature, humidity, and light intensity. This data-driven approach enhances decision-making, improves water and resource management, and increases crop yields. However, implementing WSNs in large-scale agricultural environments presents several challenges, including energy efficiency, network scalability, and reliable data transmission.

2.1 Problem Statement

One of the major issues in WSN-based smart farming is the efficient collection and transmission of data from multiple sensor nodes to a centralized Base Station (BS). Due to the limited energy resources of sensor nodes, an optimized communication protocol is required to extend network lifetime while maintaining data accuracy. Cluster-based routing is a widely used strategy to enhance network efficiency by organizing sensor nodes into clusters, where each cluster has a designated Cluster Head (CH) responsible for aggregating data and forwarding it to the BS. However, the selection and management of CHs play a crucial role in ensuring balanced energy consumption across the network.

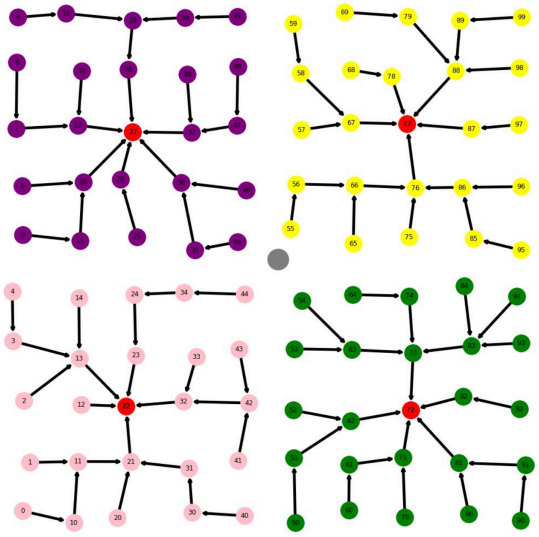
2.2 Problem Simulation

To analyze and improve the performance of WSNs in agricultural applications, this project simulates a sensor network deployment in a 400x400 field, consisting of 100 sensor nodes. The network follows a cluster-based communication approach, where node[0] serves as the Base Station (BS) and specific nodes (12, 56, 71, and 90) are assigned as Cluster Heads

(CHs). By simulating different routing strategies and communication protocols, this study aims to evaluate network performance, energy efficiency, and data reliability in an IoT-based agricultural environment. The simulation will be conducted using OMNeT++ and Castalia, providing a realistic representation of sensor node behavior under different conditions.

This project contributes to the development of optimized WSN protocols that enhance the sustainability and effectiveness of IoT applications in agriculture. The following sections will detail the system architecture, proposed protocol, simulation setup, and performance analysis.

3 Simulation Process



The simulation of our multi-cluster Wireless Sensor Network (WSN) follows a structured sequence of operations to model the real-world deployment and communication dynamics of sensor nodes (SNs) in an agricultural environment. The process consists of the following key steps:

3.1 Physical Deployment of the Network

The sensor nodes (SNs) are deployed in a **400x400** field of deployment (FoD). In practical scenarios, large-scale sensor networks in agricultural applications can be deployed by mass-dropping the SNs from a drone or aircraft over the designated area. This random distribution ensures full coverage of the field while mimicking real-world deployment strategies where manual placement of nodes may not be feasible. Each sensor node is assumed to be equipped with environmental sensors to monitor parameters such as temperature, humidity, and soil moisture.

3.2 Sink Broadcast and Cluster Head (CH) Announcement

After deployment, the Base Station (BS) (node[0]) initiates a sink broadcast to flood the network with essential information regarding the designated Cluster Heads (CHs). The selected CHs (nodes 12, 56, 71, and 90) act as intermediate relay points that manage communication within their respective clusters. The sink broadcast allows all sensor nodes to become aware of the locations of CHs, enabling them to make informed decisions about cluster membership.

3.3 Cluster Formation and Node Recruitment

Each CH enters a recruitment phase, inviting nearby sensor nodes to join their cluster. This is achieved through a controlled flooding mechanism where CHs transmit beacon signals

containing their identification and distance metrics. Sensor nodes evaluate multiple CH announcements based on parameters such as:

- **Signal Strength (RSSI)** – Nodes choose the CH with the strongest signal.
- **Distance to CH** – Nodes prefer CHs that minimize communication energy consumption.
- **Load Balancing** – Nodes consider CH capacity to avoid overloading a single CH.

Once the clusters are formed, each node communicates its data to the respective CH, which aggregates the data and transmits it to the Base Station (BS). This hierarchical communication structure improves energy efficiency by reducing the number of direct transmissions to the BS, ultimately extending the network lifetime.

4 Method Overview

To ensure efficient communication between sensor nodes in the agricultural WSN, a transport layer protocol is designed to manage data exchange and cluster-based networking operations. This protocol structures network formation and communication into three distinct phases:

4.1 Network Advertisement Phase

In this phase, the Base Station (BS) initiates the clustering process by broadcasting a network advertisement message to all nodes within the network. This message contains essential information, including the identities of pre-designated Cluster Heads (CHs). The purpose of this broadcast is to ensure that all sensor nodes in the field are aware of the available CHs, allowing them to prepare for the cluster formation process.

4.2 Cluster Recruitment Phase

Once the CHs receive the advertisement message from the BS, they proceed with a recruitment broadcast to invite surrounding sensor nodes to join their cluster. Each CH sends out a "Cluster Join Request" message, which includes:

- The CH’s node ID
- A timestamp
- The expected cluster size

Sensor nodes in the vicinity receive these messages and evaluate their options based on factors such as signal strength, distance, and energy consumption.

4.3 Cluster Joining Phase

Sensor nodes that are not designated as CHs will choose the first recruitment message they receive and respond with a "Cluster Join Acknowledgment" message, formally requesting to join that specific cluster. Once a CH receives the acknowledgment, it adds the node to its member list and assigns a unique identifier for intra-cluster communication.

5 Proposed Protocol

To establish a structured and efficient communication framework for sensor nodes, the proposed protocol defines specific message types and their respective transmission paths. The design ensures effective network formation, cluster recruitment, and node participation while maintaining energy efficiency and minimal overhead.

5.1 Packet Transmission Flow

The protocol defines three main types of control packets used during network initialization and clustering:

5.1.1 Broadcast Packet (BCAST) – Network Advertisement

- **Initiation:** The Base Station (BS) begins the network formation process by sending a BCAST packet to its immediate neighboring nodes.
- **Flooding Mechanism:**

Upon receiving a BCAST packet, a node checks if it has received a similar packet before.

If the node has not received the packet before, it stores the information about Cluster Heads (CHs) and forwards the BCAST to its neighboring nodes.

This process continues until all nodes in the Field of Deployment (FoD) receive the BCAST packet, ensuring global network awareness.

5.1.2 Recruitment Packet (RECRUIT) – Cluster Formation

- **Trigger Condition:** When a node has been designated as a Cluster Head (CH) (as indicated in the BCAST packet), it sends a RECRUIT packet to all nearby nodes, inviting them to join the cluster.
- **Joining Decision:**

Upon receiving a RECRUIT packet, a node checks whether it has already joined a cluster.

If the node has not joined any cluster yet, it accepts the invitation by responding with a JOIN packet and forwards the RECRUIT packet to other surrounding nodes.

If the node has already joined another cluster, it ignores the RECRUIT packet to prevent unnecessary energy consumption.

5.1.3 Join Packet (JOIN) – Cluster Enrollment

- **Initiation:** A sensor node that decides to join a cluster sends a JOIN packet back to the Cluster Head (CH) to formally request membership.
- **Multi-Hop Forwarding:**

If the JOIN packet does not reach the CH directly, intermediate nodes forward the packet along the path toward the CH.

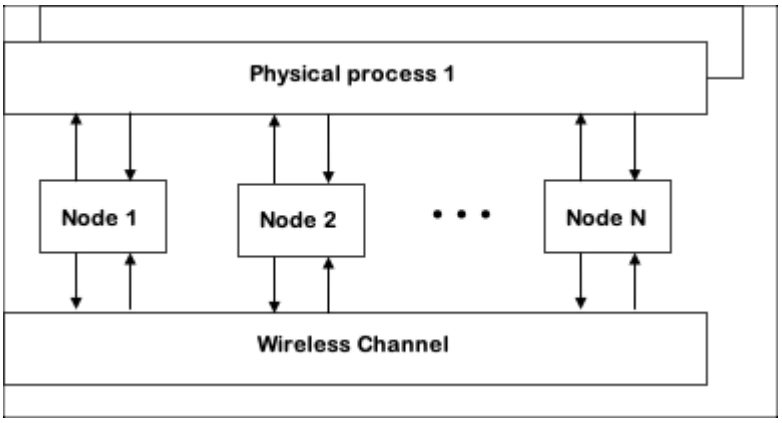
Once the CH receives the JOIN packet, it acknowledges the new member and adds the node to its cluster list.

6 Castalia Overview

Castalia is a simulation framework designed for Wireless Sensor Networks (WSN), Body Area Networks (BAN), and general wireless networks. Built on top of OMNeT++, Castalia provides a modular and flexible environment for testing communication protocols, algorithms, and network behaviors in realistic scenarios.

6.1 Structure of Castalia

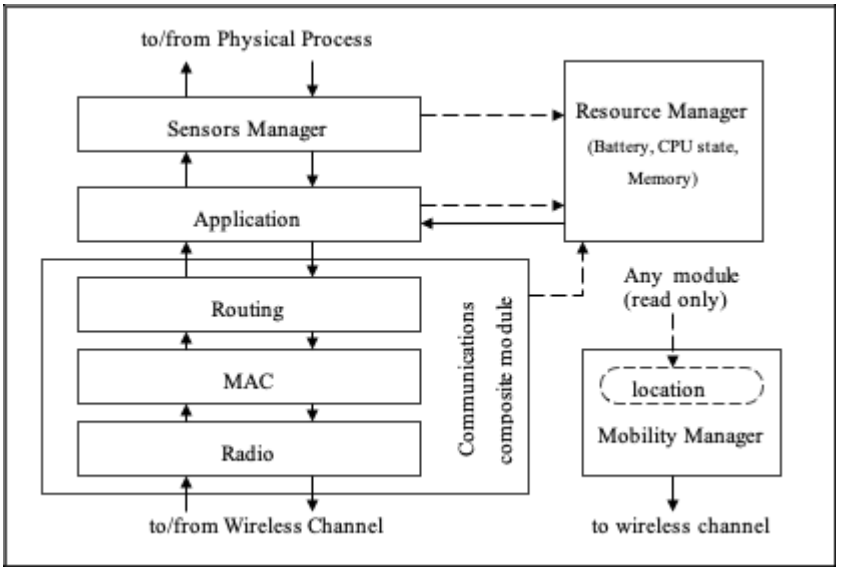
6.1.1 Modules and Connections:



Nodes (sensor devices) communicate via a wireless channel module, which handles message transmission between nodes.

Nodes also interact with physical process modules that represent environmental data (e.g., temperature, humidity). Nodes query these modules to get sensor readings.

6.1.2 Node Structure:



Each node is a composite module with submodules for resource management, application logic, MAC protocols, routing, and mobility.

Application Module: Users frequently modify this to implement custom algorithms.

MAC, Routing, Mobility Modules: These can also be changed to develop new protocols or mobility patterns.

6.1.3 Customization:

Castalia modules are highly configurable using parameters.

New protocols or applications can be built by extending existing modules or creating new ones based on abstract classes provided by Castalia.

6.1.4 Implementation:

Modules are defined using OMNeT++'s NED language (files with .ned suffix).

Simple modules are implemented in C++ (.cc and .h files).

The overall structure is defined hierarchically in .ned files and directories.

6.1.5 Run Simulation:

To start a simulation in Castalia, users navigate to **`Castalia/Simulations/radioTest/`**, where the main configuration file, `omnetpp.ini`, is located. Executing the command **`../../bin/Castalia`** lists available configurations, such as General, InterferenceTest1, and InterferenceTest2. A specific configuration can be run using the **`-c`** flag, e.g.,

../../bin/Castalia -c General

This command runs the General scenario, where a receiver (Node 0) moves through an area with two transmitters (Nodes 1 and 2). The simulation generates output files such as:

- `Castalia-Trace.txt` (event logs)
- A timestamped `.txt` file containing simulation results

By analyzing `Castalia-Trace.txt`, users can examine logs of received packets, including timestamps and module names. Other scenarios like InterferenceTest1 and InterferenceTest2 introduce mobility-based interference effects, such as collisions and signal-to-noise ratio (SINR) variations, helping users understand radio communication behaviors.

6.2 Configuration File in Castalia

Castalia uses a modular configuration approach, where each module has parameters defined in `.NED` files. The `omnetpp.ini` file plays a crucial role in defining simulation parameters such as:

- Simulation time
- Field size
- Number of nodes
- Communication protocols (MAC and Routing)
- Node mobility

The Sensor Network (SN) module manages node deployment strategies, which can be uniform, grid-based, or randomized. Wildcards allow efficient assignment of values across multiple nodes. Castalia supports multiple test scenarios within a single configuration file, enabling iterative simulations with different parameter combinations.

6.3 Modeling in Castalia

A key feature in Castalia is the `collectTraceInfo` parameter, available in all non-composite modules. By default, this parameter is set to `false`, but when enabled, it allows the module to generate trace logs stored in `Castalia-Trace.txt`.

The communication model is a core component of Castalia, encompassing aspects such as:

- Wireless channel modeling
- Radio behavior
- MAC protocol implementation

6.3.1 Wireless Channel Modeling

Castalia provides a highly realistic simulation of wireless communication, particularly for WSN and BAN applications. One crucial aspect is the average path loss modeling, which estimates signal attenuation between two nodes using the lognormal shadowing model.

$$PL(d) = PL(d_0) + 10 \cdot \eta \cdot log\left(\frac{d}{d_0}\right) + X_\sigma$$

The path loss formula is parameterized with:

- Path loss exponent ($\boldsymbol{\eta}$)
- Reference distance ($\boldsymbol{d_0}$)
- Standard deviation ($\boldsymbol{\sigma}$)

These parameters are defined in `WirelessChannel.ned` and can be accessed in `.ini` files using the prefix `SN.wirelessChannel`.

Since Castalia 3.0, the wireless channel model has been restructured. It now simply sends signal information (modulation, bandwidth, frequency, signal strength) to the radio module, which independently calculates SINR and determines packet reception.

6.3.2 Radio Module in Castalia

The radio module emulates low-power wireless radios typically used in WSNs. It includes features such as:

- Multiple operational states: transmit, receive/listen, sleep
- Transition delays between states
- Configurable transmission power levels
- Power consumption modeling
- Support for multiple modulation schemes (FSK, PSK, DiffQPSK, DiffBPSK, Ideal)
- RSSI (Received Signal Strength Indicator) calculations
- Clear Channel Assessment (CCA) capability

The Radio Parameters File defines real-world radio models like CC2420 and CC1000 from Texas Instruments. It specifies transmission power levels, noise floor, bandwidth, and sensitivity settings, allowing fine-tuned simulations.

6.3.3 MAC Protocols in Castalia

The Medium Access Control (MAC) layer significantly impacts node behavior. Castalia includes four primary MAC modules:

- **TunableMAC** – A duty-cycled MAC with extensive configuration options, including CSMA/CA behavior.
- **T-MAC** – An energy-efficient MAC using adaptive duty cycling, synchronization, and dynamic frame scheduling.
- **IEEE 802.15.4 MAC** – Standard MAC for low-power wireless networks.
- **IEEE 802.15.6 MAC** – A MAC draft proposal for Body Area Networks (BAN).

6.3.3.1 T-MAC and S-MAC

T-MAC is widely used in WSNs due to its power-saving techniques, while S-MAC is its predecessor with a more rigid duty cycle. Castalia's TMAC module supports both protocols and provides additional functionalities such as:

- Maximum transmission retries for unicast packets
- Frame synchronization and contention handling
- Timeout settings for listen periods and retransmission attempts
- RTS/CTS mechanisms for collision avoidance

These parameters are defined in TMAC.ned and can be modified in the configuration file to optimize network performance.

6.3.4 Routing

Routing in Castalia receives relatively less attention compared to other communication modules. Despite this, the routing layer offers a similar level of abstraction as other layers (such as Application and MAC), allowing users to define their own protocols. However, Castalia 3.2 provides only two built-in routing modules: `multipathRings` and `bypassRouting`. The latter, as its name suggests, does not implement any actual routing functionality.

All routing modules in Castalia share four key parameters inherited from the `iRouting.ned` interface and functionalities inherited from the `VirtualRouting` class. These parameters are:

- **collectTraceInfo** – A standard parameter found in all Castalia modules, used for trace collection.
- **maxNetFrameSize** – Determines the maximum packet size that the routing module can handle. A value of 0 or less indicates no size restriction.
- **netDataFrameOverhead** – Defines the overhead added to application packets.
- **netBufferSize** – Specifies the buffer size within the routing module.

The `bypassRouting` module does not introduce any additional parameters beyond these four.

6.4 Creating Custom Application, Routing, MAC, and Mobility Manager Modules

Directory Structure

The first step in module creation is determining the correct location within Castalia's directory structure. A dedicated directory must be created to store the source code of the new module. The appropriate directories are:

- **Application Modules:** `src/node/application/`
- **Routing Modules:** `src/node/communication/routing/`
- **MAC Modules:** `src/node/communication/mac/`
- **Mobility Manager Modules:** `src/node/mobilityManager/`

Once the directory is set up, the module must be defined using the NED language by creating a `.ned` file within the newly created directory. The file's name should match the module name.

For example, if a new MAC module named `NewCastaliaModule` is created, its corresponding directory should be named `newCastaliaModule`. Within this directory, a NED file named `NewCastaliaModule.ned` should be created.

The package of the module must be defined by replacing `/` with `.` in the relative path to Castalia's `src/` directory. For instance, if `NewCastaliaModule` is a MAC module, its package declaration in the NED file would be:

```
package node.communication.mac.newCastaliaModule;
```

Defining the Module

Each module follows a specific interface to ensure integration within Castalia. For MAC modules, the definition in `NewCastaliaModule.ned` must reference the interface module using the `like` keyword:

```
simple NewCastaliaModule like node.communication.mac.iMac
```

Parameters and Gates

Modules accept parameters at runtime via the simulation configuration. Each interface file (e.g., `iMac.ned`) defines mandatory parameters, but additional parameters can be introduced. Below is a complete example of a NED file:

```
package node.communication.mac.newCastaliaModule;

simple NewCastaliaModule like node.communication.mac.iMac {

    parameters:

        bool collectTraceInfo = default(false);

        int macMaxPacketSize = default(0);

        int macBufferSize = default(16);

        int macPacketOverhead = default(8);


        int newParameter1;
```

```
        string newParameter2 = default("default value");
        bool newParameter3 = default(false);
    gates:
        output toNetworkModule;
        output toRadioModule;
        input fromNetworkModule;
        input fromRadioModule;
        input fromCommModuleResourceMgr;
}
```

The first four parameters are mandatory and defined in `iMac.ned`. Additional parameters specific to `NewCastaliaModule` are added below them. Gates must match those in the interface file.

Implementing the Module in C++

A Castalia module is represented as a C++ class inheriting from an appropriate base class (Virtual Class). The class name must match the module name and `.ned` file name. The following base classes are used:

- **Application Modules:** `class NewCastaliaModule : public VirtualApplication`
- **Routing Modules:** `class NewCastaliaModule : public VirtualRouting`
- **MAC Modules:** `class NewCastaliaModule : public VirtualMac`
- **Mobility Manager Modules:** `class NewCastaliaModule : public VirtualMobilityManager`

The implementation files `NewCastaliaModule.h` and `NewCastaliaModule.cc` must be created. The `.cc` file must include the corresponding header file and use the `Define_Module` macro:

```
Define_Module(NewCastaliaModule);
```

Methods from the virtual class must then be implemented or overridden based on the module type.

Compiling the New Module

Once all required files are created, the following commands must be executed from the Castalia root directory to rebuild the project:

```
make clean

./makemake

make
```

For external libraries, refer to the Castalia Installation Guide for build instructions.

6.5 Defining a New Routing Module

The `VirtualRouting` class provides a foundation for implementing new routing protocols. It includes core methods for handling messages within Castalia and offers both callback methods (which must be implemented) and predefined methods (which facilitate common operations). Key callback methods include:

- **`void startup():`** Initializes protocol-specific parameters and state.
- **`void finishSpecific()`**: Called at the end of the simulation; typically not needed.
- **`void fromApplicationLayer(cPacket *pkt, const char *dstAddr):`** Handles outgoing data packets from the application layer.
- **`void fromMacLayer(cPacket *pkt, int srcMacAddress, double RSSI, double LQI)`**: Handles incoming packets from the MAC layer.
- **`void handleNetworkControlCommand(cMessage *msg)`**: Processes network control commands.
- **`void handleMacControlMessage(cMessage *msg)`**: Responds to control messages from the MAC layer.
- **`void handleRadioControlMessage(cMessage *msg)`**: Handles control messages from the radio module.

Predefined methods available within `VirtualRouting` include:

- **`void encapsulatePacket(cPacket *, cPacket *)`**: Encapsulates application packets.
- **`int bufferPacket(cPacket *pkt)`**: Stores packets in a buffer.
- **`cPacket *decapsulatePacket(cPacket *)`**: Extracts application packets.
- **`void toApplicationLayer(cMessage *msg)`**: Sends messages to the application layer.
- **`void toMacLayer(cPacket *pkt, int macAddr)`**: Sends packets to the MAC layer.
- **`int resolveNetworkAddress(const char *)`**: Converts network addresses to MAC addresses.
- **`bool isNotDuplicatePacket(cPacket *pkt)`**: Checks for duplicate packets.

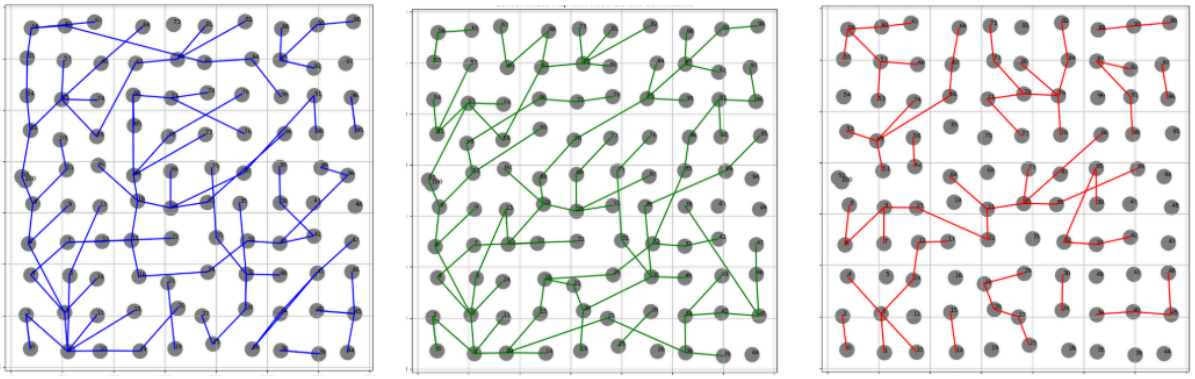
6.6 BypassRouting Module

`BypassRouting` is the simplest routing module with no multi-hop functionality. It only implements `fromApplicationLayer()` and `fromMacLayer()`.

- **`fromApplicationLayer()`**: Creates a `BypassRouting` packet, sets source and destination addresses, encapsulates the packet, and sends it to the MAC layer.
- **`fromMacLayer()`**: Checks the destination address. If it matches the current node or is a broadcast, the packet is decapsulated and sent to the application layer. Otherwise, it is discarded.

This concludes the overview of routing in Castalia and the steps to create a new module within the simulation framework.

7 Simulation Results and Evaluation



Phrase 1
99%

Phrase 2
96%

Phrase 3
78%

7.1 Broadcasting Phase

- **Packets Sent:** 338
- **Simulation Time:** 10.5s
- **Involved Nodes:** 99/100

The broadcasting phase successfully disseminated network information across the sensor field, reaching 99% of the nodes. The total 338 packets sent indicate a moderate level of network flooding, which is expected in a broadcast-based approach. The simulation time of 10.5s demonstrates an efficient spread of the base station's message, ensuring that almost all nodes become aware of the designated Cluster Heads (CHs). The 1 node that did not receive the message could be due to placement at the network boundary or temporary transmission failures.

7.2 Cluster Recruiting Phase

- **Packets Sent:** 363
- **Simulation Time:** 12.34s
- **Involved Nodes:** 96/100

During the recruitment phase, 96% of the nodes actively participated in the cluster formation process. The slightly higher number of packets (363) compared to the broadcasting phase suggests that multiple cluster heads sent recruitment messages, and some overlapping occurred. The simulation time increased to 12.34s, indicating a slightly prolonged process due to the need for cluster heads to wait for responses and recruit additional nodes. The 4 nodes that did not respond might be due to packet collisions, transmission range limitations, or delays in receiving recruitment messages.

7.3 Cluster Joining Phase

- **Packets Sent:** 60
- **Simulation Time:** 12.33s
- **Involved Nodes:** 78/100

In the final phase, 78% of the nodes successfully joined a cluster and sent a JOIN request. The low packet count (60) suggests that the JOIN messages followed efficient

multi-hop paths, reducing unnecessary transmissions. However, 22% of the nodes did not join any cluster, which could be attributed to:

- Lack of direct connectivity to a cluster head.
- Delays or packet drops during the recruitment phase.
- Nodes being too far from any cluster head's coverage range.

7.4 Overall Assessment

The simulation results indicate an effective cluster formation process with gradual reduction in packet overhead through each phase:

- Broadcasting ensures maximum reachability (99% node involvement).
- Cluster recruitment effectively engages a majority of nodes (96%).
- Cluster joining shows room for improvement, as only 78% of nodes were successfully integrated.

8 Conclusion

The proposed protocol for wireless sensor networks (WSN) in IoT-based agriculture demonstrates a structured and efficient approach to network formation. By implementing a three-phase process (Broadcasting, Cluster Recruitment, and Cluster Joining), the protocol successfully organizes sensor nodes into clusters, enabling effective environmental monitoring.

8.1 Strengths

- **Minimized packet transmission:** The protocol reduces unnecessary packet forwarding, optimizing communication overhead.
- **Fast execution time:** The network setup completes relatively quickly, ensuring timely data collection and transmission.
- **High broadcast reachability:** A significant number of nodes receive crucial network information, allowing widespread participation.

8.2 Limitations

- **Packet collisions and losses:** Due to the high number of transmissions, packets may experience collisions or losses, affecting recruitment efficiency.
- **Non-optimized routing:** The protocol does not implement advanced routing strategies, which could lead to inefficient paths and increased latency.
- **Reduced recruitment participation:** A noticeable drop in node involvement in the cluster recruitment and joining phases suggests potential coverage gaps or communication failures.

9 References

[1] Boulis, A. (2011). **Castalia User Manual** (Version 3.2). NICTA. Available at: ***https://github.com/boulis/Castalia/***