CS461 Project Report

*Zhuokun Li 134843*

# Introduction

The goal of this project is to implement the shortest path routing algorithm to guide sensing data packets from their source back to the base station through the least number of hops. A Java program is used to receive data from base station mote through serial port and display them in a user-friendly format.

# Methodology

In AS2, I have already made the JAVA program able to receive multiple sensors’ packets and display them on separate panel. Therefore, in this project, I just need to focus on implementing the shortest path algorithm.

The 1st step is to modify the Base Station program that I wrote in AS2 to make it send the PATHCALC packet in a certain period.

The 2nd step is to make sure other receiver nodes can receive this packet correctly by showing the path info such as the last hop number (the base station node id in this case), hop count (should be 1). The receiver should not send packets until receiving this packet. Then the receiver nodes should send packets to base station and the data should be displayed correctly in JAVA program.

The 3rd step is to implement the code to forward the sensor data message when a node receives from the other node.

The 4th step is to implement the code to process the PATHCALC packet it receives to find out the shortest path.

In 3rd and 4th step, the code is written but not tested because we need a very large place like the football field I used in AS3. To test the code, I need to move the node around to find out the coverage range of nodes. Therefore, before bringing the node to test, the 5th step is to make the algorithm adaptive to the movement of nodes.

The last step is designing test cases and execute them.

# Implementation Detail

## Java Program(SensorsMonitor):

This program uses the Oracle's Java Foundation Classes to build a graphical UI to show the data. The whole program contains 5 main classes:

1. SensorMonitor.java: The driver class that launch the program
2. SensorDataPanel.java: The class that represents the panel for showing each sensor’s data
3. Window.java: A class that represents the window of the program which contains a list of SensorDataPanel objects. It provides some interfaces to add and get SensorDataPanel objects.
4. DataReciever.java: The class that implements net.tinyos.message.MessageListener interface in order to receive data from Mote through serial port. It contains a list of DataSubscriber objects. Every time a new packet arrives, it will tell DataSubscriber objects to handle the data. Objects that implements DataSubscriber interface can subscribe to it.
5. ViewController.java: The class that stands between the Window and DataReceiver class. It implements the DataSubscriber interface so that when a packet is received by DataReceiver, it would tell ViewController to handle it, then ViewController would tell Window class to update the data in UI.

## BaseStation:

BaseStation uses:

* 2 SplitControl interfaces: one for serial communication with PC and the other for radio communication with other nodes. Implemented with ActiveMessageC and SerialActiveMessageC respectively.
* 2 AMSend interfaces: one for serial communication with PC and the other for sending PATHCALC message to other nodes. Implemented with SerialActiveMessageC.AMSend and lActiveMessageC.AMSend respectively.
* 1 Receive interface: for receiving sensor data from other nodes. Implemented with ActiveMessageC.Receive.
* A timer that control sending PATHCALC message in a certain period.

Time PATHCALC message is sent with the broadcast address -- AM\_BROADCAST\_ADDR, so every other node can hear this message.

## MoteView:

MoteView uses:

* 4 Read interfaces: for reading temperature, humidiy, light and voltage sensor data. Implemented with HamamatsuS10871TsrC, SensirionSht11C and DemoSensorC.
* 2 AMSend interface: one for sending sensor data packet, the other one for forwarding PATHCALC message. Implemented with ActiveMessageC.AMSend.
* 2 Receive interfaces: one for receiving sensor data packet, the other one for receiving PATHCALC message. Implemented with ActiveMessageC.Receive.
* 2 Timers: one for controlling sending sensor data in a certain period, the other for invalidating the routing information (next hop, minimum path length) in a certain period.

MoteView sends out its own sensor data packet every 1 second. Because of the split-phase operation, the reading of sensor data is completed in different time. Therefore, I use 4 bool flag to indicate if all four sensors’ data is collected or not. Every time a sensor finish reading its data, it would check if 4 flags are set to true, if yes, then it sends out the packets.

Every time it receives a sensor data packet from other nodes, it checks if the radio control is busy, if not, it forwards the packet.

Every time it receives a PATHCALC packet, it follows the Shortest Path Algorithm to determine whether to forward it, whether to update its own routing table.

## LED Signal:

BaseStation:

* Every time it sends out a PATHCALC message, it blinks R, G, B lights.
* Every time it receives a MoteView message (containing sensor data) from other nodes, it blinks G, B lights.

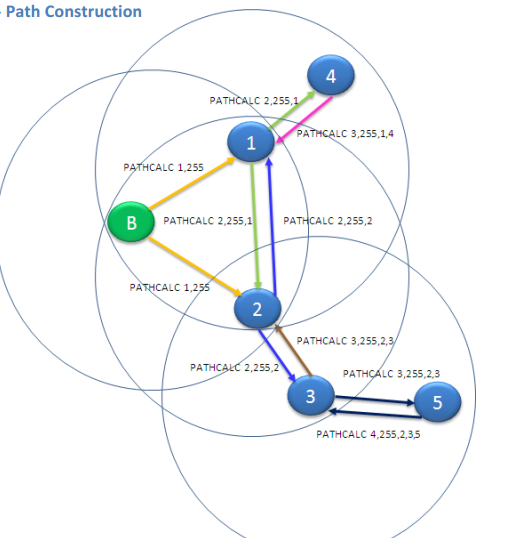
MoteView:

* Every time it sends out a MoteView message (containing sensor data), it blinks the LED lights with the id of node it is sending to which is the next hop number.
* Since it would invalidate its routing table, the next hop will be set to -1 and it will blink G, B lights.

The BaseStation is set to #7 and the other 5 nodes are set to # 1-5 so when a sender node shows its next hop is #6, we know that its routing table is invalidated.

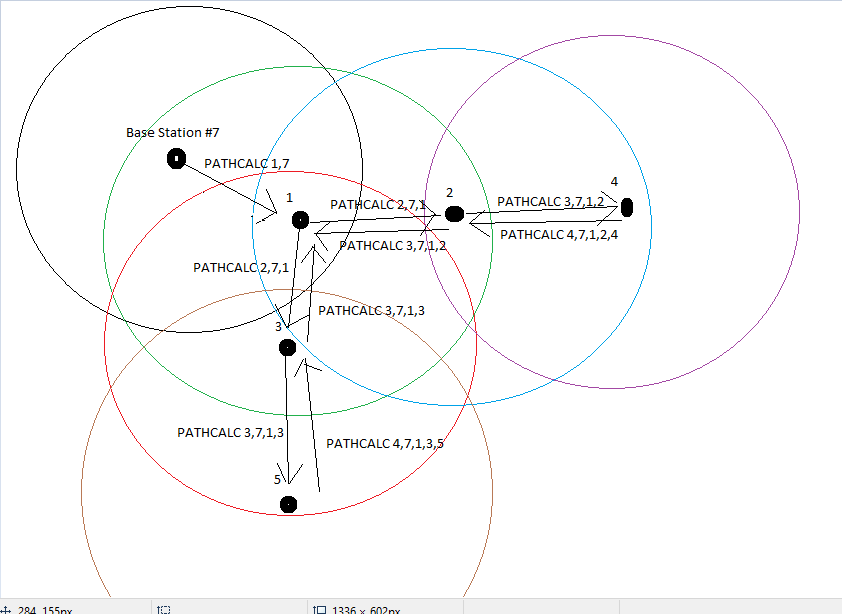
# Network Topology Diagram

I tested with 2 topology diagrams, the first one is the same as the one in the project requirement:



*Figure 1 Topology Diagram 1*

The other one is a little bit different:



*Figure 2 Topology Diagram 2*

# Problems & Challenges

The 1st challenge is finding a proper place for experiment since the radio range of nodes are very big even I use the lowest power level.

The 2nd problem is that the result of testing is not very ideal. A node cannot always receive PATHCALC packet successfully to update its routing information. Say, in the 2nd topology diagram above, node 2 and 3 should be able to receive #1’s PATHCALC message to update its routing table. But in fact, whenever #2 and #3 invalidate their routing message and wait for a new PATHCALC message from #1, it takes a long time to successfully receive a the PATHCALC message and update their next hop to 1. When I move #2 and #3 closer to #1, there’s a higher rate for them to receive PATHCALC message but they would also receive the PATHCALC message from BaseStation (#7) and then send their packet directly to BaseStation but not through #1. Similarly, the sensor data message sometimes cannot be received successfully by the next hop.

# Conclusion

1. The original Shortest Path Algorithm is not adaptive to mobile sensor nodes. A simple solution is invalidating the routing table in a certain period so that nodes can update their routing table. This also helps debugging when moving a node further and further from another node to test the coverage range.
2. The coverage range is not so ideally a regular circle as topology diagram shows. It may be influenced by real environment and changes with time. When a node A can receive message from node B, it is not guaranteed that B can receive from A since their coverage range is different. The reception rate decreases with distance and the relationship is not linear so it is hard to model the range ideally.
3. The result of testing Shortest Path Algorithm is more close to expectation when nodes’ distances maintains in a certain period. Say in diagram 2, if #2 and #3 are in the effective communication range of #1 (reception rate > 90%) and in the Poor communication range of Base Station (reception rate < 30%), most of time node 2 and node 3 would choose node 1 as their next hop and the result is closer to expectation.