LAB 3: Routing and Flooding

ENGG4200

Instructor:
Dr.Petros Spachos

Mon-10:30 Section

Date: 10th of November, 2020

Bilal Ayyache: 0988616

Harmeet Singh: 1027662

Contents

1 Introduction **1**

2 Grid Topology **1**

3 Random Path **2**

A Appendix **3**

A.0.1 Part 1 Code 3
A.0.2 Part 2 Code 6
A.0.3 Network 1 Setup 10
A.0.4 Network 2 Setup 11

List of Figures

1 Simulation Output- Grid Topology 1
2 Simulation Output- Random Path 2
3 Grid Topology Network 10
4 Grid Topology Network code 10
5 Random Path Network 11
6 Random Path Network code 12

1 Introduction

This lab was divided into 2 parts, the first part focuses on building a grid topology and finding the routing between source and destination to successfully transfer messages. For the second part we learn how to flood a message in the WSN and make use of relay nodes to achieve this objective. Both parts require a grid topology network layout and Omnet++ was used to successfully complete the lab.

2 Grid Topology

```
PC[0] generated 10 received 1 -- Transmitted 10
PC[1] generated 0 received 5 -- Transmitted 5
PC[2] generated 0 received 5 -- Transmitted 5
PC[3] generated 0 received 5 -- Transmitted 5
PC[4] generated 0 received 5 -- Transmitted 5
PC[5] generated 0 received 5 -- Transmitted 5
PC[6] generated 0 received 0 -- Transmitted 0
PC[7] generated 0 received 0 -- Transmitted 0
PC[8] generated 0 received 0 -- Transmitted 0
PC[9] generated 0 received 5 -- Transmitted 5
PC[10] generated 0 received 5 -- Transmitted 5
PC[11] generated 0 received 0 -- Transmitted 0
PC[12] generated 0 received 0 -- Transmitted 0
PC[13] generated 0 received 0 -- Transmitted 0
PC[14] generated 0 received 5 -- Transmitted 5
PC[15] generated 0 received 5 -- Transmitted 5
PC[16] generated 0 received 0 -- Transmitted 0
PC[17] generated 0 received 0 -- Transmitted 0
PC[18] generated 0 received 0 -- Transmitted 0
PC[19] generated 0 received 5 -- Transmitted 5
PC[20] generated 0 received 5 -- Transmitted 5
PC[21] generated 0 received 0 -- Transmitted 0
PC[22] generated 0 received 0 -- Transmitted 0
PC[23] generated 0 received 0 -- Transmitted 0
PC[24] generated 0 received 5 -- Transmitted 5
PC[25] generated 0 received 5 -- Transmitted 5
PC[26] generated 0 received 5 -- Transmitted 5
PC[27] generated 0 received 5 -- Transmitted 5
PC[28] generated 0 received 5 -- Transmitted 5
PC[29] generated 0 received 10 -- Transmitted 0
```

Figure 1: Simulation Output- Grid Topology

For the grid topology 30 sensors were connected using four gates. These gates are up, down, left, and right. In initialization, 10 different messages where created with different schedule time. The source, destination, and message ID was set in these messages. The source of the message is always node 0. The destination of the message is always the last node. In the grid topology the last node is node 29. The message ID was used to determine the path of the current message. This ID was the number of the message transmitted. Referring to the code in the appendix section A.0.1,

a message is only generated in the initialization function. The handle message checks if message was received by the last node. If the message is received, the message is deleted. If the message is not at destination, it gets forwarded. This function counts the received messages in each node. After handling the message, the message gets forwarded to the path determined by the message ID. If message ID is even, it takes the first route. If the message ID is odd, it takes the second route. Using the send function, the network can successfully send the message from the first node to the last node. Information about the nodes are later printed using the finish function.

3 Random Path

```
PC[75] generated 0 received 2 -- Transmitted 2
PC[76] generated 0 received 2 -- Transmitted 2
PC[77] generated 0 received 1 -- Transmitted 1
PC[78] generated 0 received 1 -- Transmitted 1
PC[79] generated 0 received 8 -- Transmitted 8
PC[80] generated 0 received 0 -- Transmitted 0
PC[81] generated 0 received 0 -- Transmitted 0
PC[82] generated 0 received 0 -- Transmitted 0
PC[83] generated 0 received 0 -- Transmitted 0
PC[84] generated 0 received 0 -- Transmitted 0
PC[85] generated 0 received 0 -- Transmitted 0
PC[86] generated 0 received 2 -- Transmitted 2
PC[87] generated 0 received 1 -- Transmitted 1
PC[88] generated 0 received 1 -- Transmitted 1
PC[89] generated 0 received 9 -- Transmitted 9
PC[90] generated 0 received 0 -- Transmitted 0
PC[91] generated 0 received 0 -- Transmitted 0
PC[92] generated 0 received 0 -- Transmitted 0
PC[93] generated 0 received 0 -- Transmitted 0
PC[94] generated 0 received 0 -- Transmitted 0
PC[95] generated 0 received 0 -- Transmitted 0
PC[96] generated 0 received 0 -- Transmitted 0
PC[97] generated 0 received 1 -- Transmitted 1
PC[98] generated 0 received 1 -- Transmitted 1
PC[99] generated 0 received 10 -- Transmitted 0
```

Figure 2: Simulation Output- Random Path

In the random path exercise, a fully connected network with 8 gates per node was created. These gates are up, down, left, right, and 4 diagonal gates. This simulation was very similar to the grid topology exercise. The only changes were implemented in forwarding the message. A random number was used to calculate a random path. due to the fact that there are only 3 gates needed to go from the top left to bottom right, there are 720 different routes that can be taken. These gates are right, down, and diagonal down right.

A Appendix

A.0.1 Part 1 Code

```
#include "node.h"

Define_Module(Node);



int pathCounter = 0;
int generatedinSource = 0;

 void Node::initialize()
 {
     int src = 0;
     int dest = 29;
     timeout = 1.0;

     char msgname[20];
     sprintf(msgname, "Msg-%d-to-%d", src,dest);

     if(getIndex() == 0){
         for(int i = 0; i<10;i++){
             char msgname[20];
             //-------Generating First Message-----------
             NodeMsg* msg = new NodeMsg(msgname);
             msg -> setSource(src);
             msg -> setDestination(dest);
             msg -> setMessageID(i);
             generated++;
             //----------------------------------------
```

```
            sprintf(msgname, "Node-%d", getIndex());
            cMessage *mymsg = check_and_cast<cMessage *>(msg);

            scheduleAt(simTime()+timeout,mymsg);
            timeout = timeout + 1.0;
        }
    }
}


void Node::handleMessage(cMessage *msg)
{
    NodeMsg *mymsg = check_and_cast<NodeMsg *>(msg);

    if(getIndex() == 29){
        delete(msg);
        recieved++;
    }else{
        if(getIndex() != 0 ){
            pathCounter++;
            recieved++;
        }
        forwardMessage(mymsg);
    }
}

void Node::forwardMessage(cMessage*msg)
{
    int messageID = 0;

    EV<< "Forwarding Message " << msg <<" on port out[" << 29 << "]\n";
```

```
    transmitted++;

    NodeMsg *mymsg = check_and_cast<NodeMsg *>(msg);
    mymsg->setHopCount(mymsg->getHopCount()+1);

    messageID = mymsg->getMessageID();

    //PATH

    if(messageID % 2 == 0){
        if(this->gate("right$o")->isConnected()){
            send(mymsg,"right$o");
        }else if(this->gate("down$o")->isConnected()){
            send(mymsg, "down$o");
        }
    }else{
        if(this->gate("down$o")->isConnected()){
            send(mymsg,"down$o");
        }else if(this->gate("right$o")->isConnected()){
            send(mymsg, "right$o");
        }
    }
}

void Node::finish()
{
    printf("PC[%d] generated %d received %d -- Transmitted %d\n", getIndex(), generated, re
}
```

A.0.2 Part 2 Code

```
#include "node.h"

Define_Module(Node);


int pathCounter = 0;
int generatedinSource = 0;


 void Node::initialize()
 {
     int src = 0;
     int dest = 29;
     timeout = 1.0;

     char msgname[20];
     sprintf(msgname, "Msg-%d-to-%d", src,dest);

     if(getIndex() == 0){
         recieved++;
         for(int i = 0; i<10;i++){
             char msgname[20];
             //-------Generating First Message-----------
             NodeMsg* msg = new NodeMsg(msgname);
             msg -> setSource(src);
             msg -> setDestination(dest);
             msg -> setMessageID(i);
             generated++;
             //----------------------------------------
```

```
            sprintf(msgname, "Node-%d", getIndex());
            cMessage *mymsg = check_and_cast<cMessage *>(msg);

            scheduleAt(simTime()+timeout,mymsg);
            timeout = timeout + 1.0;
        }
    }
}


void Node::handleMessage(cMessage *msg)
{
    NodeMsg *mymsg = check_and_cast<NodeMsg *>(msg);

    if(getIndex() == 99){
        delete(msg);
        recieved++;
    }else{
        if(getIndex() != 0 ){
            pathCounter++;
            recieved++;
        }
        forwardMessage(mymsg);
    }
}

void Node::forwardMessage(cMessage*msg)
{
    int messageID = 0;
    int option = intuniform(0,3);
```

```
EV<< "Forwarding Message " << msg <<" on port out[" << 29 << "]\n";
transmitted++;

NodeMsg *mymsg = check_and_cast<NodeMsg *>(msg);
mymsg->setHopCount(mymsg->getHopCount()+1);

messageID = mymsg->getMessageID();


//PATH
switch(option){
case 0:
    if(this->gate("right$o")->isConnected()){
        send(mymsg,"right$o");
    }else if(this->gate("downright$o")->isConnected()){
        send(mymsg, "downright$o");
    }else if(this->gate("down$o")->isConnected()){
        send(mymsg, "down$o");
    }
    break;
case 1:
    if(this->gate("right$o")->isConnected()){
        send(mymsg,"right$o");
    }else if(this->gate("down$o")->isConnected()){
        send(mymsg, "down$o");
    }else if(this->gate("downright$o")->isConnected()){
         send(mymsg, "downright$o");
    }
    break;
case 2:
    if(this->gate("down$o")->isConnected()){
```

```
            send(mymsg,"down$o");
        }else if(this->gate("right$o")->isConnected()){
            send(mymsg, "right$o");
        }else if(this->gate("downright$o")->isConnected()){
            send(mymsg, "downright$o");
        }
        break;
    case 3:
        if(this->gate("downright$o")->isConnected()){
            send(mymsg,"downright$o");
        }else if(this->gate("down$o")->isConnected()){
            send(mymsg, "down$o");
        }else if(this->gate("right$o")->isConnected()){
            send(mymsg, "right$o");
        }
        break;
    }

}

void Node::finish()
{
    printf("PC[%d] generated %d received %d -- Transmitted %d\n", getIndex(), generated, re
}
```

A.0.3 Network 1 Setup

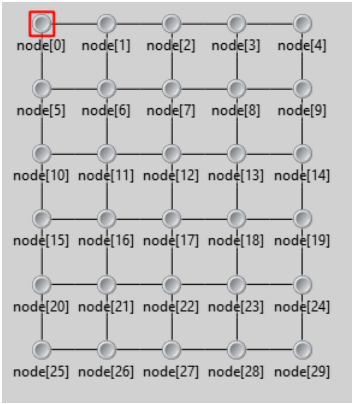

Figure 3: Grid Topology Network

```
network Network
{
    parameters:
        int height @prompt("Number of rows") = default(6);
        int width @prompt("Number of columns") = default(5);
        @display("bgb=366,264");
    submodules:
        node[height*width]: GridNode {
            parameters:
                @display("p=,,m,$width");
        }
    connections allowunconnected:
        for i=0..height-1, for j=0..width-1 {
            node[i*width+j].down <--> node[(i+1)*width+j].up if i!=height-1;
            node[i*width+j].right <--> node[(i*width+j)+1].left if j!=width-1;


        }
}
```

Figure 4: Grid Topology Network code

A.0.4 Network 2 Setup

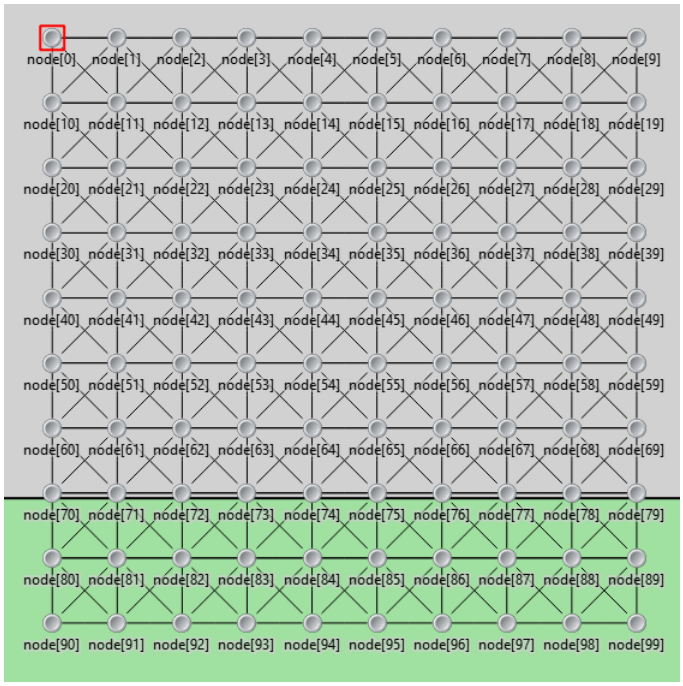

Figure 5: Random Path Network

```
simple GridNode extends Node
{
    gates:
        inout up;
        inout left;
        inout down;
        inout right;
        inout downleft;
        inout downright;
        inout upleft;
        inout upright;
}


network Network
{
    parameters:
        int height @prompt("Number of rows") = default(10);
        int width @prompt("Number of columns") = default(10);
        @display("bgb=366,264");
    submodules:
        node[height*width]: GridNode {
            parameters:
                @display("p=,,m,$width");
        }
    connections allowunconnected:
        for i=0..height-1, for j=0..width-1 {
            node[i*width+j].down <--> node[(i+1)*width+j].up if i!=height-1;
            node[i*width+j].right <--> node[(i*width+j)+1].left if j!=width-1;

            node[i*width+j].downright <--> node[(i*width+j+11)].upleft if i!=height-1 && j!=width-1;
            node[i*width+j].downleft <--> node[(i*width+j+9)].upright if i!=height-1 && j!=width;

        }
}
```

Figure 6: Random Path Network code