İSTANBUL TEKNİK ÜNİVERSİTESİ

1773

BLG 433E

COMPUTER COMMUNICATIONS

CRN: 12337

PROJECT #2

Submission Date: 03.12.2014

GROUP MEMBERS:

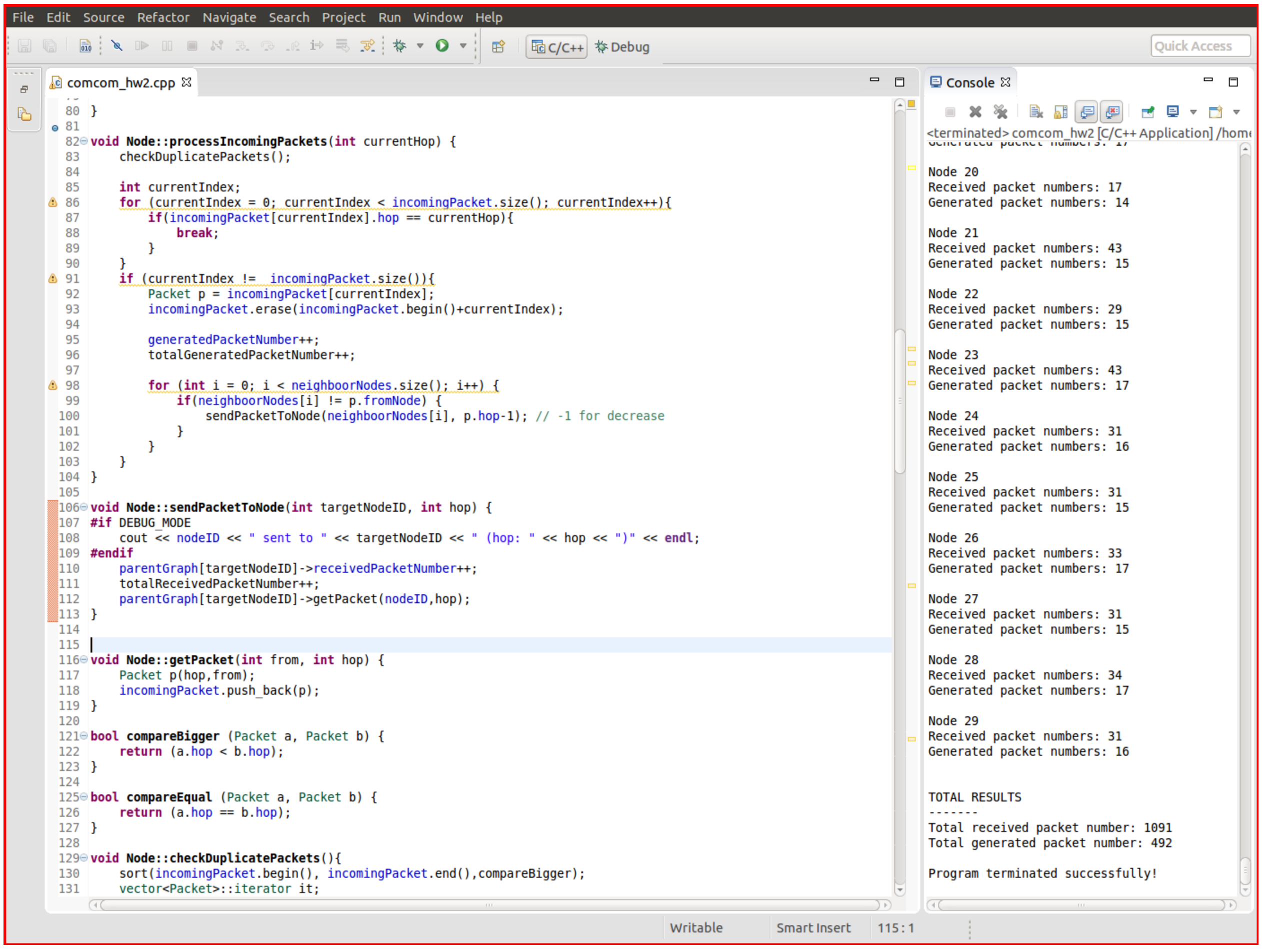
TUĞRUL YATAĞAN	040100117
EMRE GÖKREM	040100124

A. Introduction

In this project a random graph is created and then flooding algorithm runs on this graph.

B. Development Environment, Compilation and Running

All implementation is developed under Ubuntu Linux via Eclipse integrated development environment.

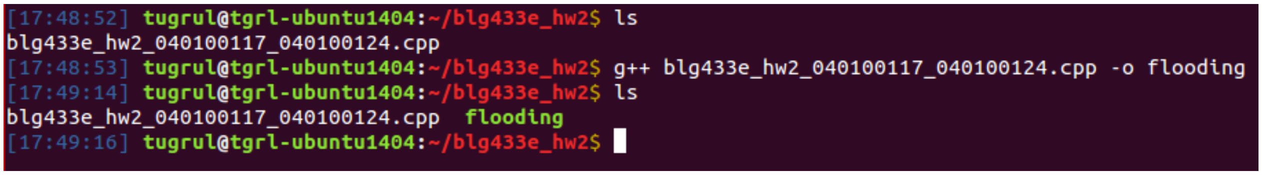


```cpp
}

void Node::processIncomingPackets(int currentHop) {
    checkDuplicatePackets();

    int currentIndex;
    for (currentIndex = 0; currentIndex < incomingPacket.size(); currentIndex++){
        if(incomingPacket[currentIndex].hop == currentHop){
            break;
        }
    }
    if (currentIndex !=  incomingPacket.size()){
        Packet p = incomingPacket[currentIndex];
        incomingPacket.erase(incomingPacket.begin()+currentIndex);

        generatedPacketNumber++;
        totalGeneratedPacketNumber++;

        for (int i = 0; i < neighboorNodes.size(); i++) {
            if(neighboorNodes[i] != p.fromNode) {
                sendPacketToNode(neighboorNodes[i], p.hop-1); // -1 for decrease
            }
        }
    }
}

void Node::sendPacketToNode(int targetNodeID, int hop) {
#if DEBUG_MODE
    cout << nodeID << " sent to " << targetNodeID << " (hop: " << hop << ")" << endl;
#endif
    parentGraph[targetNodeID]->receivedPacketNumber++;
    totalReceivedPacketNumber++;
    parentGraph[targetNodeID]->getPacket(nodeID,hop);
}


void Node::getPacket(int from, int hop) {
    Packet p(hop,from);
    incomingPacket.push_back(p);
}

bool compareBigger (Packet a, Packet b) {
    return (a.hop < b.hop);
}

bool compareEqual (Packet a, Packet b) {
    return (a.hop == b.hop);
}

void Node::checkDuplicatePackets(){
    sort(incomingPacket.begin(), incomingPacket.end(),compareBigger);
    vector<Packet>::iterator it;
```

```
Generated packet numbers: 17

Node 20
Received packet numbers: 17
Generated packet numbers: 14

Node 21
Received packet numbers: 43
Generated packet numbers: 15

Node 22
Received packet numbers: 29
Generated packet numbers: 15

Node 23
Received packet numbers: 43
Generated packet numbers: 17

Node 24
Received packet numbers: 31
Generated packet numbers: 16

Node 25
Received packet numbers: 31
Generated packet numbers: 15

Node 26
Received packet numbers: 33
Generated packet numbers: 17

Node 27
Received packet numbers: 31
Generated packet numbers: 15

Node 28
Received packet numbers: 34
Generated packet numbers: 17

Node 29
Received packet numbers: 31
Generated packet numbers: 16


TOTAL RESULTS
-------
Total received packet number: 1091
Total generated packet number: 492

Program terminated successfully!
```

All the code is in one .cpp file. The application can be compiled with:

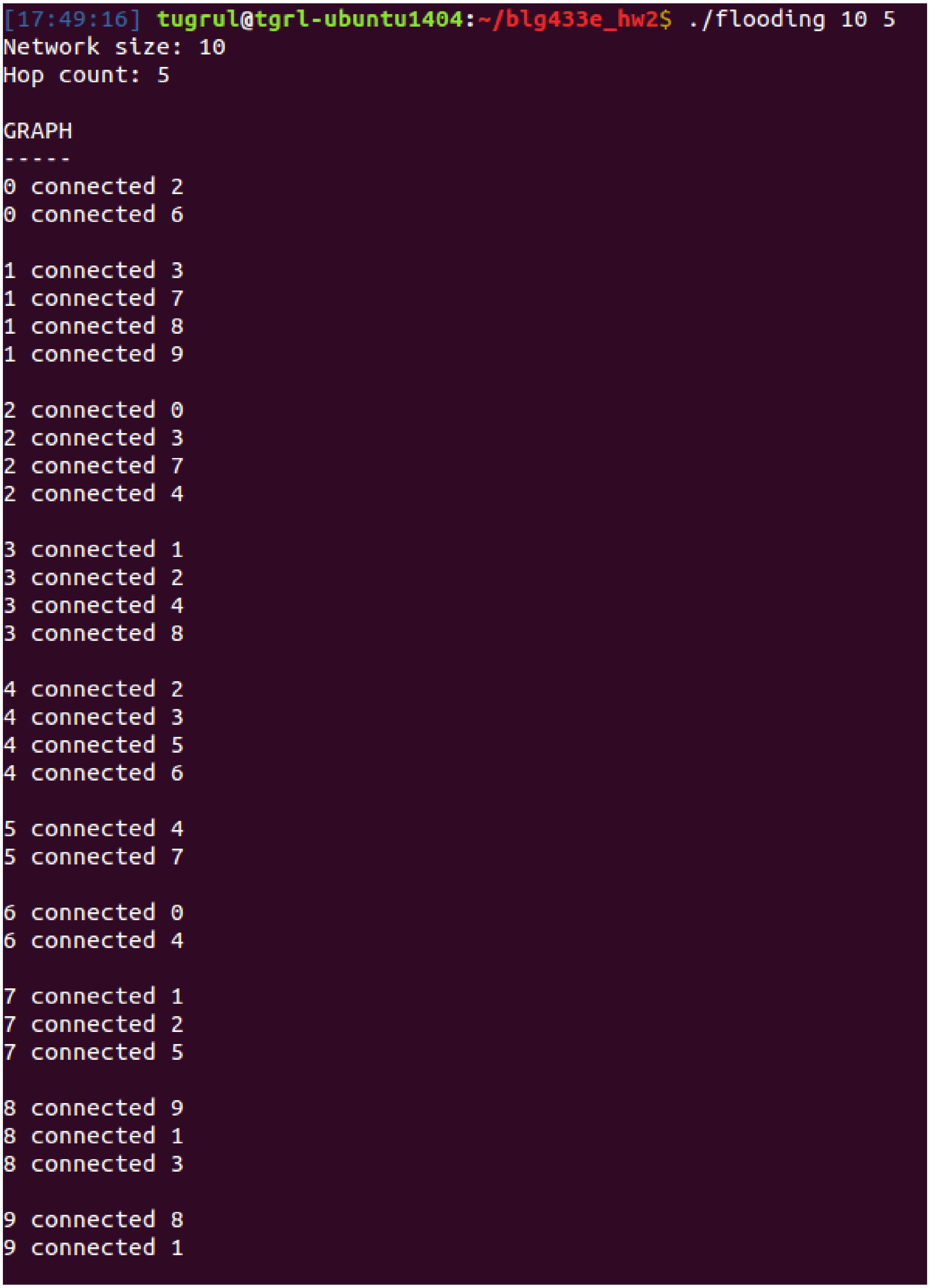
```
g++ blg433e_hw2_040100117_040100124.cpp -o flooding
```



```
[17:48:52] tugrul@tgrl-ubuntu1404:~/blg433e_hw2$ ls
blg433e_hw2_040100117_040100124.cpp
[17:48:53] tugrul@tgrl-ubuntu1404:~/blg433e_hw2$ g++ blg433e_hw2_040100117_040100124.cpp -o flooding
[17:49:14] tugrul@tgrl-ubuntu1404:~/blg433e_hw2$ ls
blg433e_hw2_040100117_040100124.cpp  flooding
[17:49:16] tugrul@tgrl-ubuntu1404:~/blg433e_hw2$ 
```

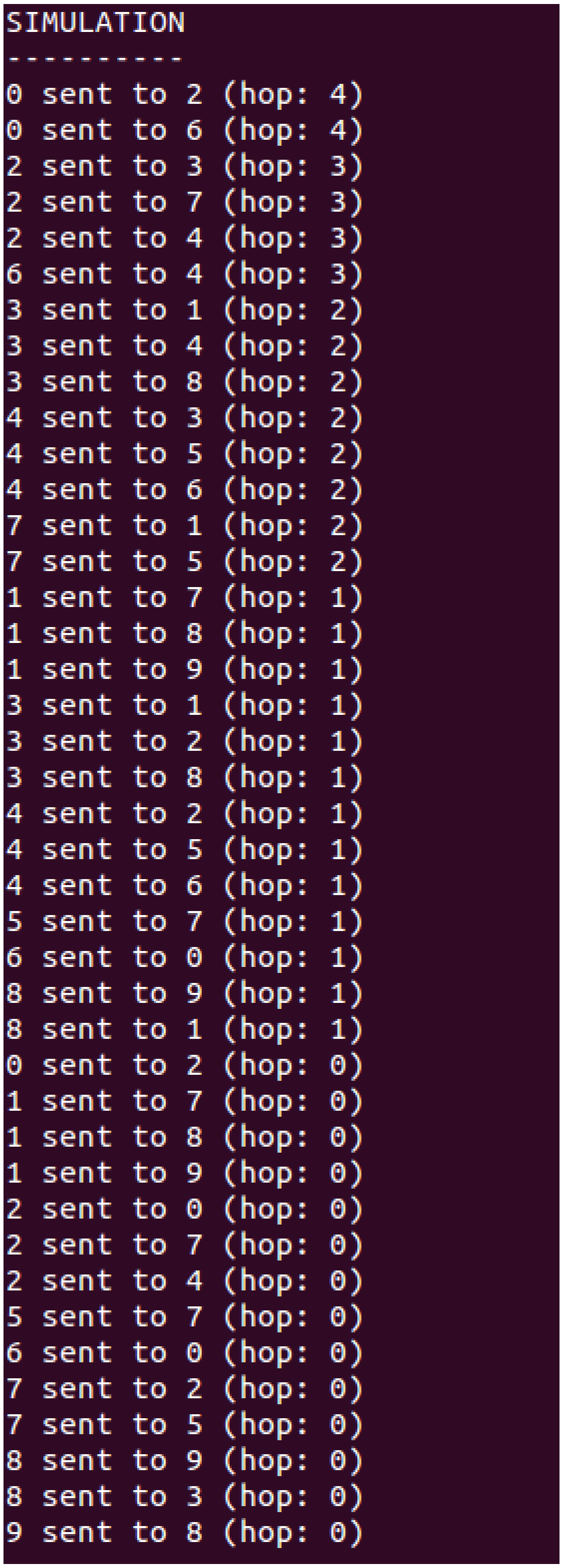
And binary application can be run with:

```
    ./flooding network_size hop_count
```



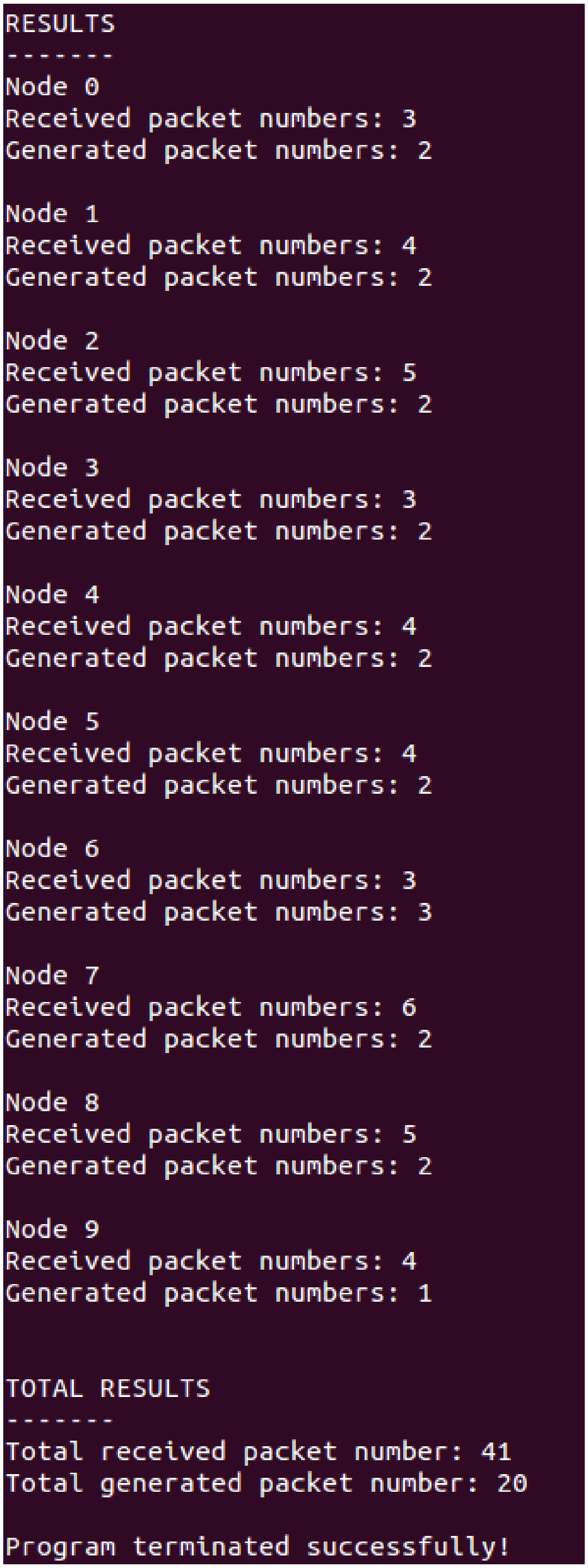
```
[17:49:16] tugrul@tgrl-ubuntu1404:~/blg433e_hw2$ ./flooding 10 5
Network size: 10
Hop count: 5

GRAPH
-----
0 connected 2
0 connected 6

1 connected 3
1 connected 7
1 connected 8
1 connected 9

2 connected 0
2 connected 3
2 connected 7
2 connected 4

3 connected 1
3 connected 2
3 connected 4
3 connected 8

4 connected 2
4 connected 3
4 connected 5
4 connected 6

5 connected 4
5 connected 7

6 connected 0
6 connected 4

7 connected 1
7 connected 2
7 connected 5

8 connected 9
8 connected 1
8 connected 3

9 connected 8
9 connected 1
```

Firstly randomly generated graph’s connection map is shown at top and then simulation process steps are shown.



```
SIMULATION
----------
0 sent to 2 (hop: 4)
0 sent to 6 (hop: 4)
2 sent to 3 (hop: 3)
2 sent to 7 (hop: 3)
2 sent to 4 (hop: 3)
6 sent to 4 (hop: 3)
3 sent to 1 (hop: 2)
3 sent to 4 (hop: 2)
3 sent to 8 (hop: 2)
4 sent to 3 (hop: 2)
4 sent to 5 (hop: 2)
4 sent to 6 (hop: 2)
7 sent to 1 (hop: 2)
7 sent to 5 (hop: 2)
1 sent to 7 (hop: 1)
1 sent to 8 (hop: 1)
1 sent to 9 (hop: 1)
3 sent to 1 (hop: 1)
3 sent to 2 (hop: 1)
3 sent to 8 (hop: 1)
4 sent to 2 (hop: 1)
4 sent to 5 (hop: 1)
4 sent to 6 (hop: 1)
5 sent to 7 (hop: 1)
6 sent to 0 (hop: 1)
8 sent to 9 (hop: 1)
8 sent to 1 (hop: 1)
0 sent to 2 (hop: 0)
1 sent to 7 (hop: 0)
1 sent to 8 (hop: 0)
1 sent to 9 (hop: 0)
2 sent to 0 (hop: 0)
2 sent to 7 (hop: 0)
2 sent to 4 (hop: 0)
5 sent to 7 (hop: 0)
6 sent to 0 (hop: 0)
7 sent to 2 (hop: 0)
7 sent to 5 (hop: 0)
8 sent to 9 (hop: 0)
8 sent to 3 (hop: 0)
9 sent to 8 (hop: 0)
```

Lastly results per node and results per total are shown at the end of the program.



```
RESULTS
-------
Node 0
Received packet numbers: 3
Generated packet numbers: 2

Node 1
Received packet numbers: 4
Generated packet numbers: 2

Node 2
Received packet numbers: 5
Generated packet numbers: 2

Node 3
Received packet numbers: 3
Generated packet numbers: 2

Node 4
Received packet numbers: 4
Generated packet numbers: 2

Node 5
Received packet numbers: 4
Generated packet numbers: 2

Node 6
Received packet numbers: 3
Generated packet numbers: 3

Node 7
Received packet numbers: 6
Generated packet numbers: 2

Node 8
Received packet numbers: 5
Generated packet numbers: 2

Node 9
Received packet numbers: 4
Generated packet numbers: 1


TOTAL RESULTS
-------
Total received packet number: 41
Total generated packet number: 20

Program terminated successfully!
```

C. Answers for Report Questions

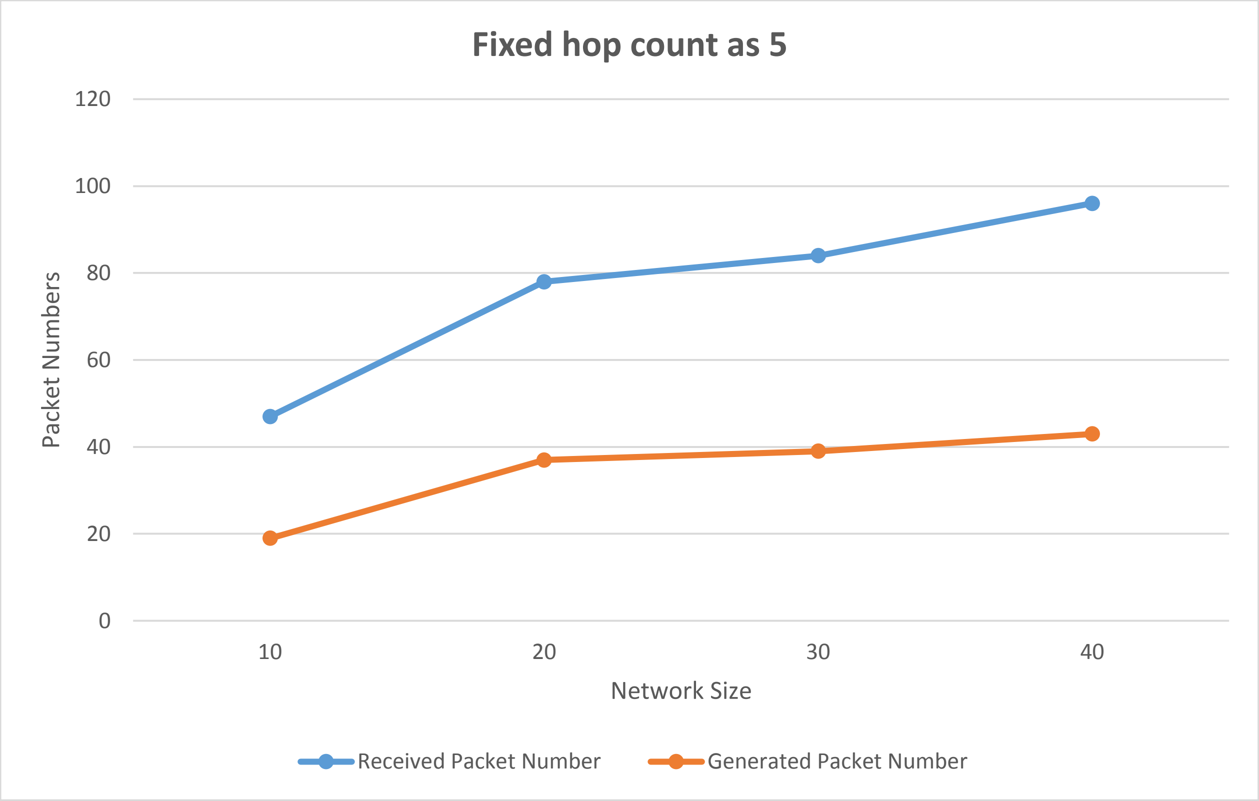
1. Creating network graph

- `Graph` class is responsible for creating network graph with connecting `Node` classes together. At initialization, a random degree between 2 and 4 assigns at every node. And then `Graph::createGraph()` function connects nodes randomly node by node. Every node keeps its connection in `vector<int> neighboorNodes` vector. If previously assigned node degrees are not sufficient, suitable nodes degree's incremented by `Graph::incrementSuitableDegree()` function. If node's degree is already 4 (maximum), node's degree cannot incremented. After all nodes connected, `Graph::checkConnected()` check if all nodes are connected by BFS traversing network with `Graph::traverseGraph(int id)` recursive function. If graph is not connected `Graph::reconnectGraph()` function cleared old graph connection map and recreates graph until the graph is confirmed as connected.
- Every instance of `Node` class has `vector<Packet>incomingPacket` vector to hold incoming packets. If a node sends packet to another node, an instance of `Packet` class is pushed target node's incoming packet vector. `Node::processIncomingPackets()` function of every node process packet sending steps in `Graph::startSimulation()` function within a loop, node by node. Simulation runs as hop count number for every node, if node has packet to proceed, node sends its received packet to all its neighbor nodes.
- Every node sends packet to neighboring nodes incoming packet vector so at the same time target node gets it receiving packet. Whole network scanned in a loop for if node has a receiving packet. If node receives packet(s), node eliminates duplicate packets and send a newly generated packet with decreased hop count to all its neighboring nodes. All nodes are scanned for number of hop count in main simulation loop. Received and generated packet counters keeps track of every nodes statistic during simulation and after simulation total received and generated packet counter statistics are created.

2. Simulation with hop count as 5

Received and generated packet numbers table for **fixed hop count as 5**:

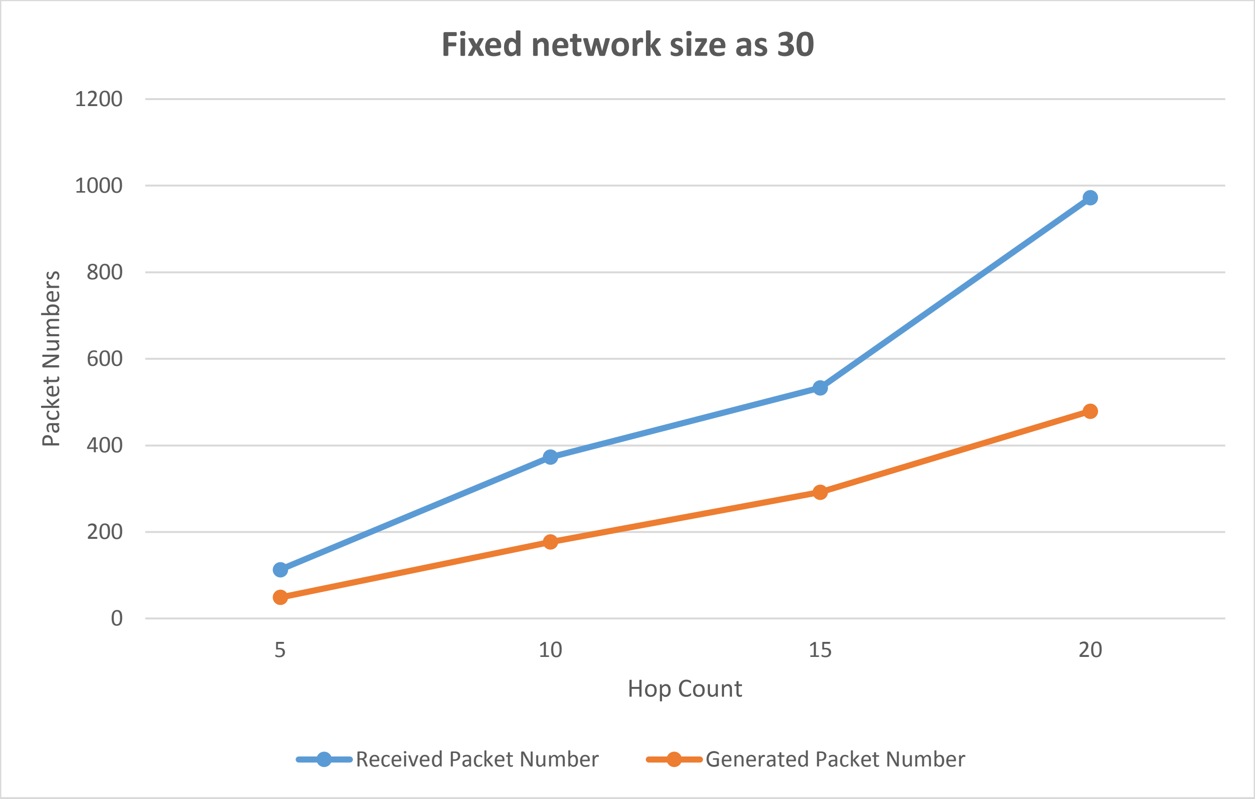
Simulation results for fixed hop count as 5		
Network Size	Received Packet Number	Generated Packet Number
10	47	19
20	78	37
30	84	39
40	96	43



3. Simulation with network size as 30

Received and generated packet numbers table for **fixed network size as 30**:

Simulation results for fixed network size as 30		
Hop Count	Received Packet Number	Generated Packet Number
5	113	49
10	373	177
15	533	292
20	972	479



4. Explanation

The number of generated packed increases with both of hop count and network size increase. Generated number depends on connections of nodes so sending packet between nodes. If network size increases, number of nodes increases. Because of this, the number of packet sending increases therefore the number of generated packets increases. Likewise, if the hop count increases, the number of packet sending increases therefore the number of generated packets increases. But effect of the hop count is bigger than effect of the network size because the hop count provides sending more packets at same time. If the hop counter is bigger, more nodes generate packets.