NETWORKS PROGRAMMING ASSIGNMENT

FLOOD ROUTING SIMULATION & SOLUTIONS TO OVERCOME LOOPING

106119055 | CSE-A

Jeremiah Thomas

Do note:-

- I have recorded a video, to explain some concepts that I used but found hard to put in a written format. Kindly visit this link for the same: https://www.youtube.com/watch?v=5KKkni3nSSE
- Kindly find all the relevant source codes attached in the zip folder
- Do ensure that C++11 support is there for codeConcurrent.cpp

---

My Approach:-

The same flood routing algorithm can be achieved using various logicS. But in my approach, I saw a very similar pattern to BFS and Flooding. Thus, I decided to take a BFS approach, **keeping the core of BFS** but **modifying it extensively to reflect Flooding.**

Also, if you notice, you would see 2 code files: **concurrent** and **nonconcurrent**. I wanted to reflect a real time flooding network, and so went with the concurrent approach of having multiple threads to handle sending different packets simultaneously.

However, with the use of threads, come an issue: While the simulation is accurate to the dot, it is hard for a viewer to understand the simulation as the threads output into one another.

And that is why I have thought it best to retain the nonConcurrent approach.

Also, do note that there are **2 levels** of concurreny:

Level 1 -> wherein the packets are all sent at the same time on different threads but concurrent flooding of a particular packet from a source does not occur.

Level 2 -> Here both are ensured. Packets are sent/received concurrently while also flooding from a source concurrently.

Now, for my simulation, I have some essential class types & entities that are common to both the concurrent and nonconcurrent approach:-

Packet class:-

```
class packet
{
    public:
    int src;
    int dest;
    int seq_no;
    double ttl_left = 5;    //time to live
    double startTime;
    unordered_set<int> nodesVisited;
    int maxhopCount = networkDiameter;
    int maxRetransmit = 3;
};
```

src -> It is the source node from which packet will be sent

dest -> Destination node for a packet

seq_no -> Sequence Number assigned to each packet in the order they are processed

ttl_left -> Time To live left

startTime -> A timestamp to record when a packet was sent initially

nodesVisited -> Used to keep track of all the nodes a certain packet visited. Return true/false in O(1)

maxHopCount -> Maximum number of hops permitted for a packet. Decrements with each hop.

maxRetransmit -> Maximum no of retransmits permitted for a packet once it has been dropped. Decrements each time.

Node class:-

```
class node
{
    public:
    vector<int> nbrs;
    unordered_set<int> floodedPackets;
    int id;

};
```

nbrs -> A dynamic array to hold the neighbours of current node

floodedPackets -> An unordered set to keep track of all the packets the current node has previously flooded

Other important Entities :-

```
node nn[1000];
packet pkt[1000];
bool reached[1000] = { false
};

bool dropped[1000] = { false
};
```

nn -> An array , Network Nodes of node type to hold node related info

pkt -> Ar array of packet type to hold info regarding all pkts

reached -> An array of acknowledgement from receiver that a packet has reached

dropped -> An array of acknowledgment from receiver that a packet has been dropped.

Core Functions:-

routingIndividualPkt(packet pkt):-

This function takes care of the routing of each individual packet. It runs a while loop until acknowledge is received that either the packet has been received or has been dropped.

In the loop, it timestamps the packet for its startTime and runs our modified bfs algo on the packet.

During the run of bfs algo, we will either get a drop or reached ack. If reached, all good.

```
void routingIndividualPkt(packet &pkt)
{
    retransmit:
    while (!reached[pkt.seq_no] &&
!dropped[pkt.seq_no])
    {
        //set startTime for curPacket
        std::chrono::time_point<std::chrono::system
_clock > curTime =
std::chrono::system_clock::now();

        std::chrono::duration<double>
elapsed_seconds = curTime - startSimTime;

        //std::time_t end_time =
std::chrono::system_clock::to_time_t(endSimTime);

        pkt.startTime = elapsed_seconds.count();
```

```
        cout << "\n\nSending Packet:" << pkt.seq_no
<< " at: " << pkt.startTime << "s\n";

        bfs(pkt.src, pkt.dest, pkt.seq_no);

        if (reached[pkt.seq_no])
        {
            cout << "\nPacket:"<<pkt.seq_no<<" has
been successfully received!\n";
        }

    }
```

However, if dropped, then we retransmit the algorithm on receiving the drop acknowledgement

```
    if(dropped[pkt.seq_no]){
        //retransmit after sometime
        if(pkt.maxRetransmit--){
        cout<<"\nRetransmitting
Packet:"<<pkt.seq_no;
        goto retransmit;
    }else{
        cout<<"\nRetransmits for
Packet:"<<pkt.seq_no<<" exhausted!\n";
    }
    }
}
```

In the concurrent approach, there is a slight addition. Since we use threads, we have the extra privilege of putting the thread to sleep.

So here, after a second, the current thread resumes:-

```
    if(dropped[pkt.seq_no]){
    //Make the current thread sleep for a second
before retransmitting...
    std::this_thread::sleep_for
(std::chrono::seconds(1));
        //retransmit after sometime
        if(pkt.maxRetransmit--){
        cout<<"\nRetransmitting
Packet:"<<pkt.seq_no;
        goto retransmit;
    }
```

bfs():-

I've tried to write comments wherever necessary to explain the different nuances.

So in a nutshell, I use a queue to store the neighbours of the current node. I start the flooding process from the src node.

```
void bfs(int src, int dest, int seq_no)
{
    queue<int> q;

    q.push(src);
    pkt[seq_no].nodesVisited.insert(src);
```

Then , I run a while loop until the queue is empty and has nothing more to flood.

```
    while (!q.empty())
    {
```

Whenever I pop a node from the queue, it is symbolic of the popped node receiving the packet referenced by seq_no in argument list.

```
//Here it is meant that seq_no has now been received at
curN
        int curN = q.front();   //curN -> current Node
        q.pop();

//check if current node is the destination
        if (curN == dest)
        {
            reached[seq_no] = true;
            cout << "Destination reached (Node:" << dest
<< ") \n";
            break;
        }

 //check if packet has exceeded its time to live factor:-
```

Here I check if the packet has exceeded it TTL by comparing the start timepoint of simulation with the packet's own start Timestamp. If its greater than TTl, then the packet is dropped.

```
std::chrono::time_point<std::chrono::system_clock >
curTime = std::chrono::system_clock::now();

        std::chrono::duration<double> elapsed_seconds =
curTime - startSimTime;

        std::time_t curTimePrint =
std::chrono::system_clock::to_time_t(curTime);
```

```
        float curTimeInSeconds = elapsed_seconds.count();

        cout << "Packet: " << seq_no << ", Received at
Node: " << curN << " at : " << curTimeInSeconds << "s" <<
endl;

        if (pkt[seq_no].ttl_left < (curTimeInSeconds -
pkt[seq_no].startTime))
        {
            //Drop packet
            cout << "Dropping Packet due to exceeding TTL
\n";
            dropped[seq_no] = true;
            break;
        }

 //check if hopCount has exhausted to zero, if so drop the
packet
Here with each new node visited, the hop count of the pkt
decreases
        if (--pkt[seq_no].maxhopCount == 0)
        {
            //Drop packet
            cout << "Dropping Packet due to exhausting
HopCount \n";
            dropped[seq_no] = true;
            break;
        }

 //Check if the packet has been already flooded by curN or
not
```

This is one of other looping solutions to prevent a node from flooding the same packet again by storing every flooded node in the unordered set floodedPackets and reviewing it was flooded or not.

If the find() function return an end character, then it is clear that it a new packet for the current node. So, flood

```
 if (nn[curN].floodedPackets.find(seq_no) ==
nn[curN].floodedPackets.end())
        {

            nn[curN].floodedPackets.insert(seq_no);
//Insert into set

 //Flooding the packet to nbr nodes on conditions


  cout << "From Node: " << curN << ", we are ... \n";

            for (auto i: nn[curN].nbrs)
            {
                //  cout<<i<<"\t";
       if (pkt[seq_no].nodesVisited.find(i) ==
pkt[seq_no].nodesVisited.end())
                {

//this packet has not visited node "i" yet
        q.push(i);
        cout << "Flooding to Node:" << i << endl;
        pkt[seq_no].nodesVisited.insert(i);
//create a new thread and run bfs(i, dest, seq_no)
                }
                else
                {
```

```
//packet visited node i before, so not visiting
cout << "Sequence no:" << seq_no << " has visited Node:"
<< i << " before" << endl;
                }
            }

            cout << endl;

        }
        else
        {
            //already flooded, so ignore
            cout << "Already Flooded! \n";
        }
    }
}
```

The bfs approach stays the same for Level 1 of concurrency but for Level 2,

(Found in bfs2 for routingIndividualPkt_lvl2)

Here, a new array of threads is created to handle flooding for a particular referenced by seq_no in the argument list

```
thread *arr = new thread[nn[curN].nbrs.size()];
//creating threads for concurrent execution
cout << "From Node: " << curN << ", we are ... \n";

            for (auto i: nn[curN].nbrs)
            {
                //  cout<<i<<"\t";
                if
(pkt[seq_no].nodesVisited.find(i) ==
pkt[seq_no].nodesVisited.end())
```

```
                {
//this packet has not visited node "i" yet
 q.push(i);
 cout << "Flooding to Node:" << i << endl;
pkt[seq_no].nodesVisited.insert(i);
```

And for each unique flood to another node, the array of thread is assigned to a new process of running bfs2 function from updated source node = i

```
 //create a new thread and run bfs(i, dest,seq_no)
 arr[i] = thread(bfs2, i, dest, seq_no);
                }
```

Now for Concurrent Levels:-

In the main() fn, the calling is different to ensure packets are concurrently sent/received:-

Here for every new packet, a thread is allocated from noTransmission[] array.

```
    for(int i=0;i < 3; i++){

    cout<<"\nAssigning new Thread for Packet:"<<i<<" ....
\n";
```

Based on the level of concurrency chosen by user, the thread is given bfs1 or bfs2 fns to execute

```
    if(levelConcurrency ==1){
    noTransmissions[i] = thread(routingIndividualPkt_lvl1,
pkt[i]);
}else{
```

```
    noTransmissions[i] = thread(routingIndividualPkt_lvl2,
pkt[i]);
}

    }
```

We then join each thread to ensure each packet transmission is done

```
    for(int i=0;i < 3;i++){

    cout<<"\nJoining Thread of Packet : "<<i<<"\n";
    noTransmissions[i].join();

    }
```

User Interactivity & I/O Setting:-

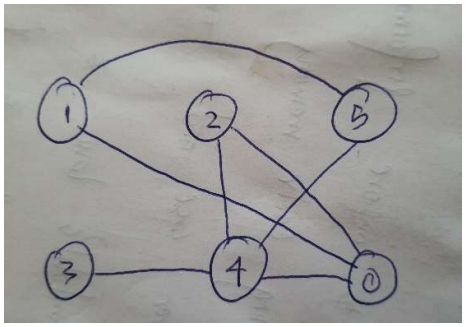
I've tried to keep it as interactive as possible with error correction and validation at each step to ensure no flawed data is entered.

I've included comments and kept the print stmts clear so that its easy for a reader to understand

Also, the user has the option to go with a prebuilt network for convenience's sake.

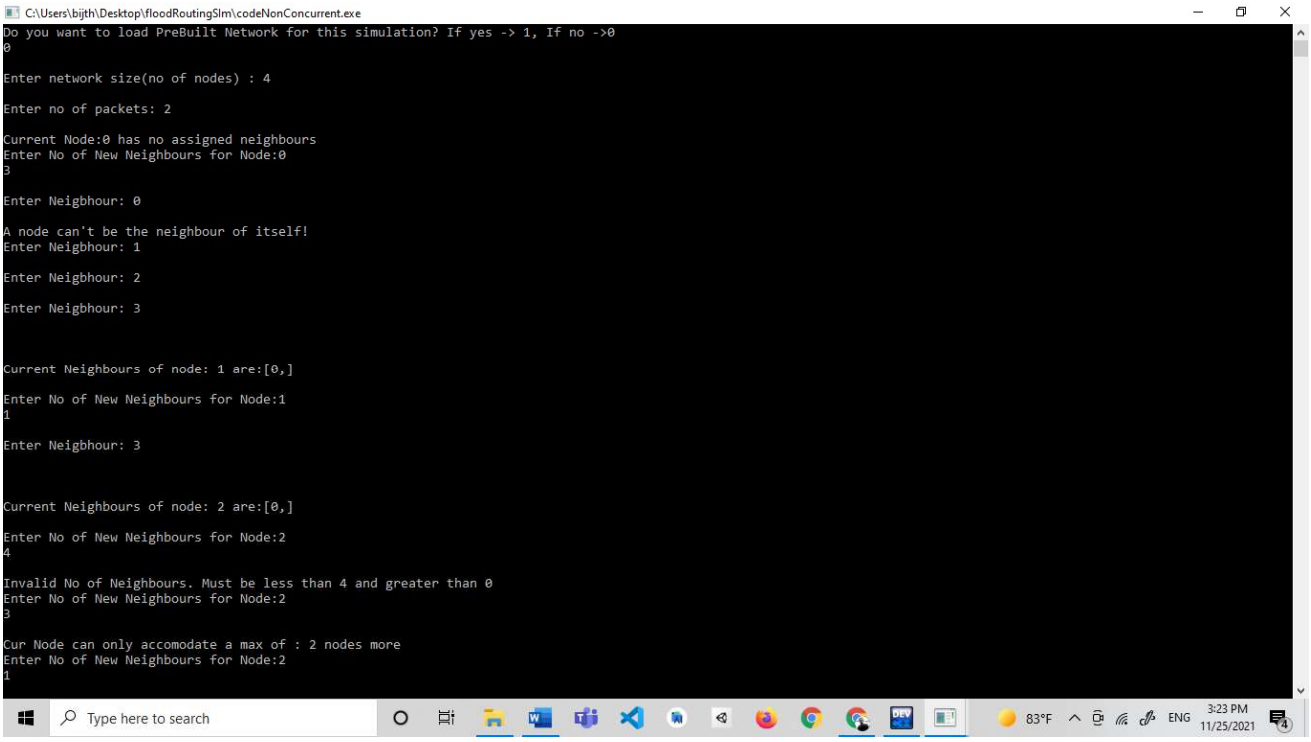
(Do Scroll Down)

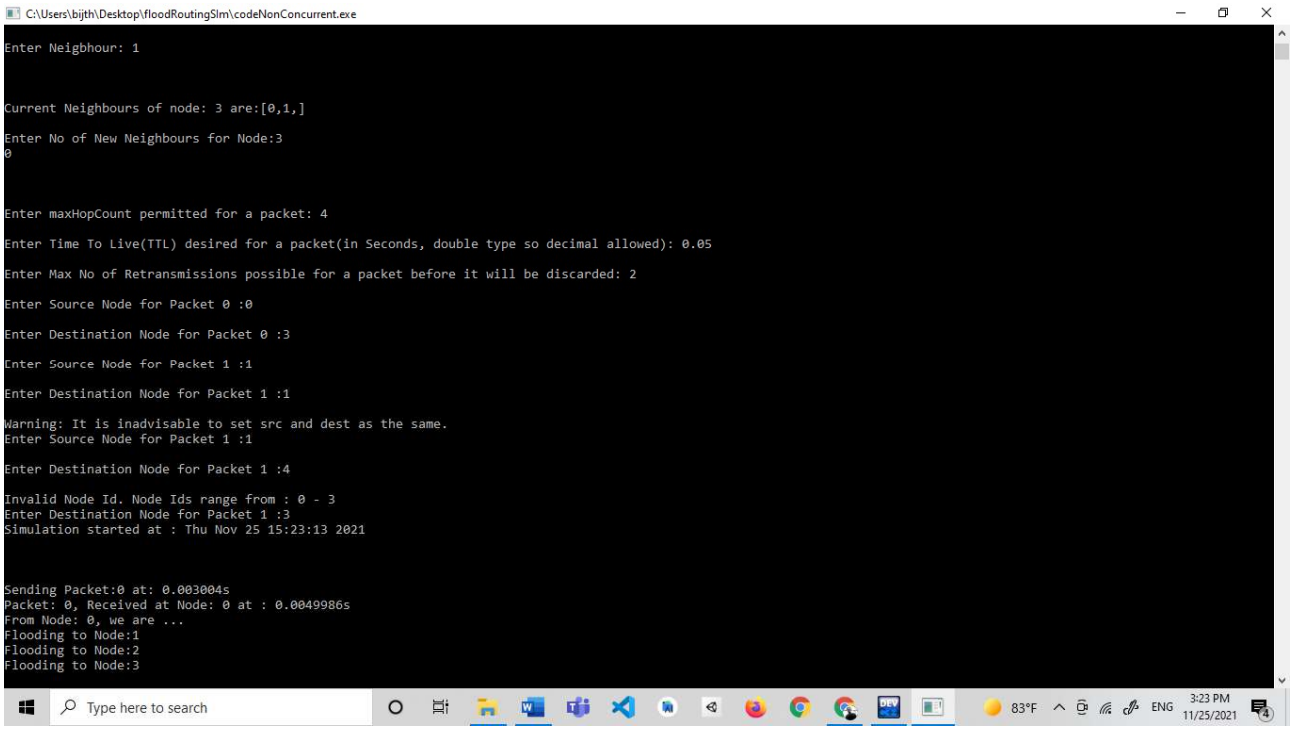
For the screenshots of the below cases which you will see as you scroll down, I took the below prebuilt network setting: -

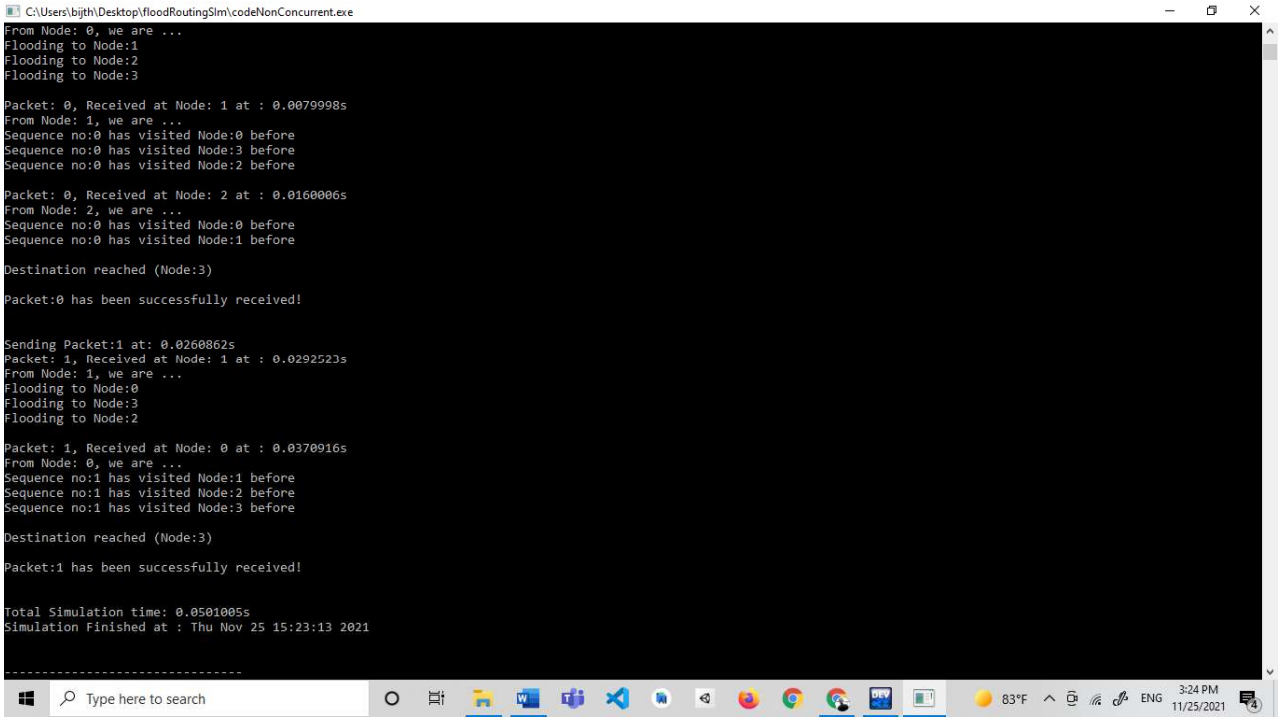


Seq No	Src	Dest
0	0	5
1	1	4
2	2	3

However, if a user is going the Interactive route, it will look something like this:-



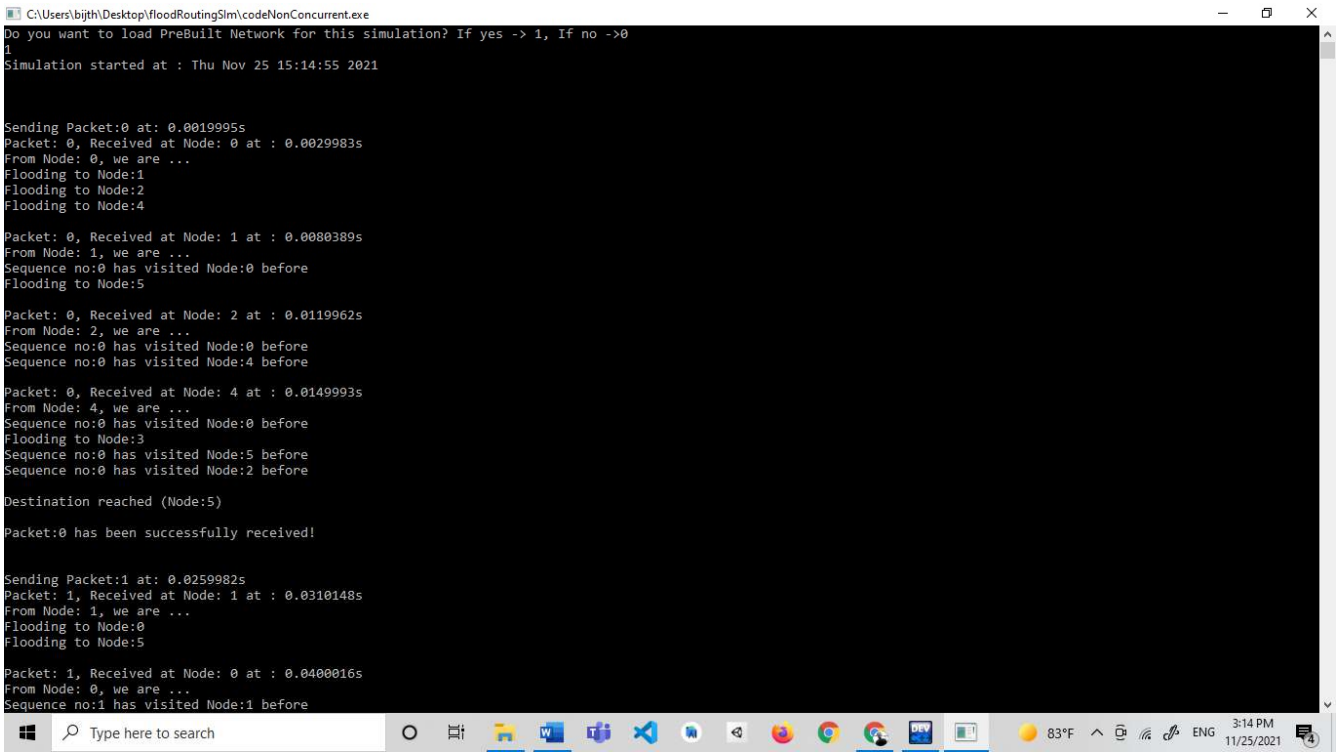

C:\Users\bijth\Desktop\floodRoutingSim\codeNonConcurrent.exe
Enter Neigbhour: 1
Current Neighbours of node: 3 are:[0,1,]
Enter No of New Neighbours for Node:3
0
Enter maxHopCount permitted for a packet: 4
Enter Time To Live(TTL) desired for a packet(in Seconds, double type so decimal allowed): 0.05
Enter Max No of Retransmissions possible for a packet before it will be discarded: 2
Enter Source Node for Packet 0 :0
Enter Destination Node for Packet 0 :3
Enter Source Node for Packet 1 :1
Enter Destination Node for Packet 1 :1
Warning: It is inadvisable to set src and dest as the same.
Enter Source Node for Packet 1 :1
Enter Destination Node for Packet 1 :4
Invalid Node Id. Node Ids range from : 0 - 3
Enter Destination Node for Packet 1 :3
Simulation started at : Thu Nov 25 15:23:13 2021
Sending Packet:0 at: 0.003004s
Packet: 0, Received at Node: 0 at : 0.0049986s
From Node: 0, we are ...
Flooding to Node:1
Flooding to Node:2
Flooding to Node:3
Type here to search
83°F
ENG
3:23 PM
11/25/2021


C:\Users\bijth\Desktop\floodRoutingSim\codeNonConcurrent.exe
From Node: 0, we are ...
Flooding to Node:1
Flooding to Node:2
Flooding to Node:3
Packet: 0, Received at Node: 1 at : 0.0079998s
From Node: 1, we are ...
Sequence no:0 has visited Node:0 before
Sequence no:0 has visited Node:3 before
Sequence no:0 has visited Node:2 before
Packet: 0, Received at Node: 2 at : 0.0160006s
From Node: 2, we are ...
Sequence no:0 has visited Node:0 before
Sequence no:0 has visited Node:1 before
Destination reached (Node:3)
Packet:0 has been successfully received!
Sending Packet:1 at: 0.0260862s
Packet: 1, Received at Node: 1 at : 0.0292523s
From Node: 1, we are ...
Flooding to Node:0
Flooding to Node:3
Flooding to Node:2
Packet: 1, Received at Node: 0 at : 0.0370916s
From Node: 0, we are ...
Sequence no:1 has visited Node:1 before
Sequence no:1 has visited Node:2 before
Sequence no:1 has visited Node:3 before
Destination reached (Node:3)
Packet:1 has been successfully received!
Total Simulation time: 0.0501005s
Simulation Finished at : Thu Nov 25 15:23:13 2021
Type here to search
83°F
ENG
3:24 PM
11/25/2021

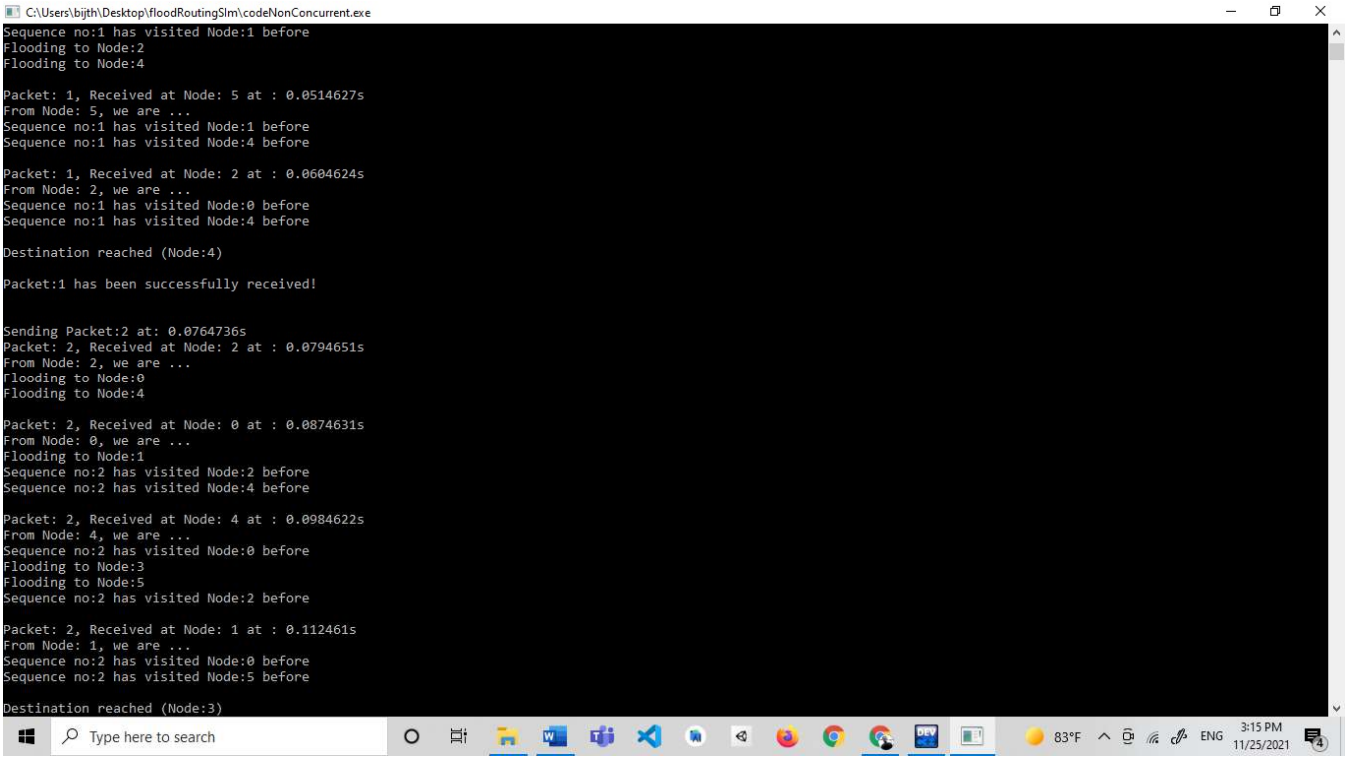
Sample Output Screenshots for Success:-

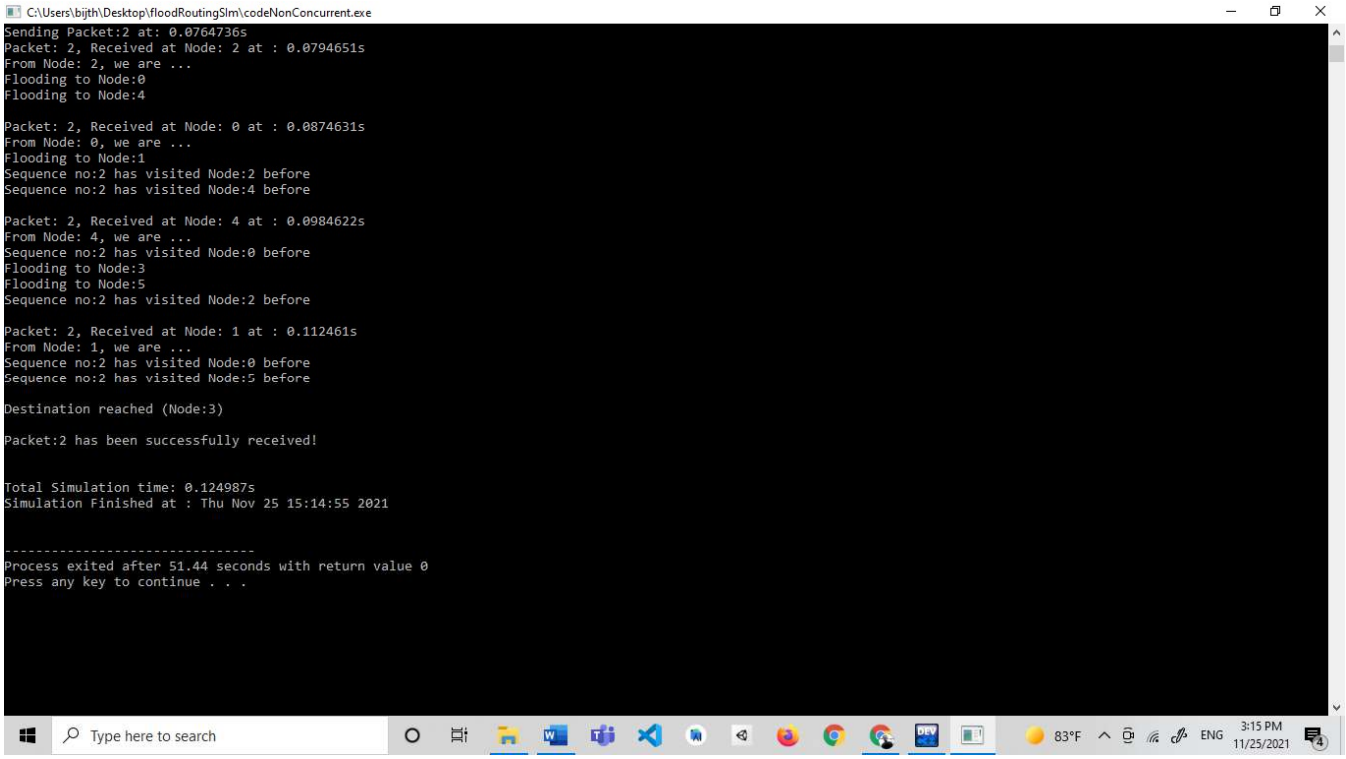
(Took liberty on maxHopCount (=10)and TTL so that all packets woul successfully reach destination)

nonConcurrent:-



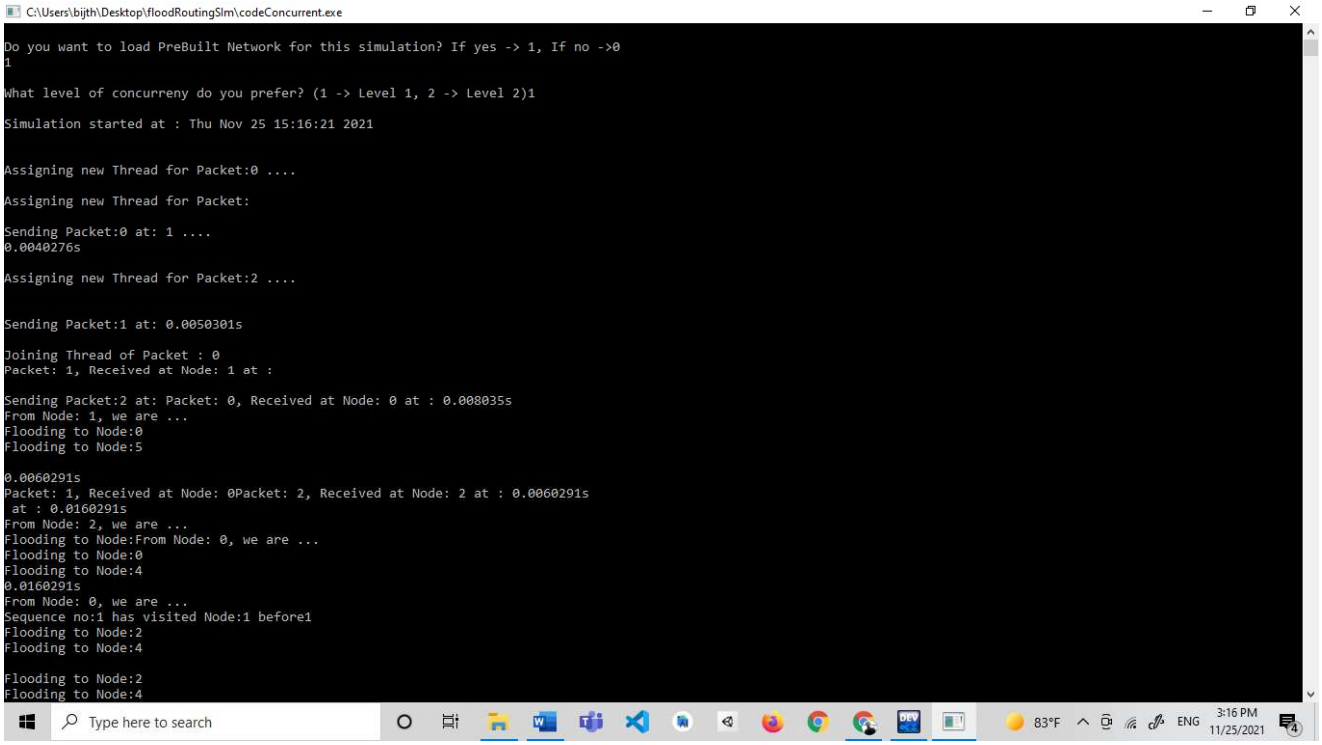
(Do Scroll Down)

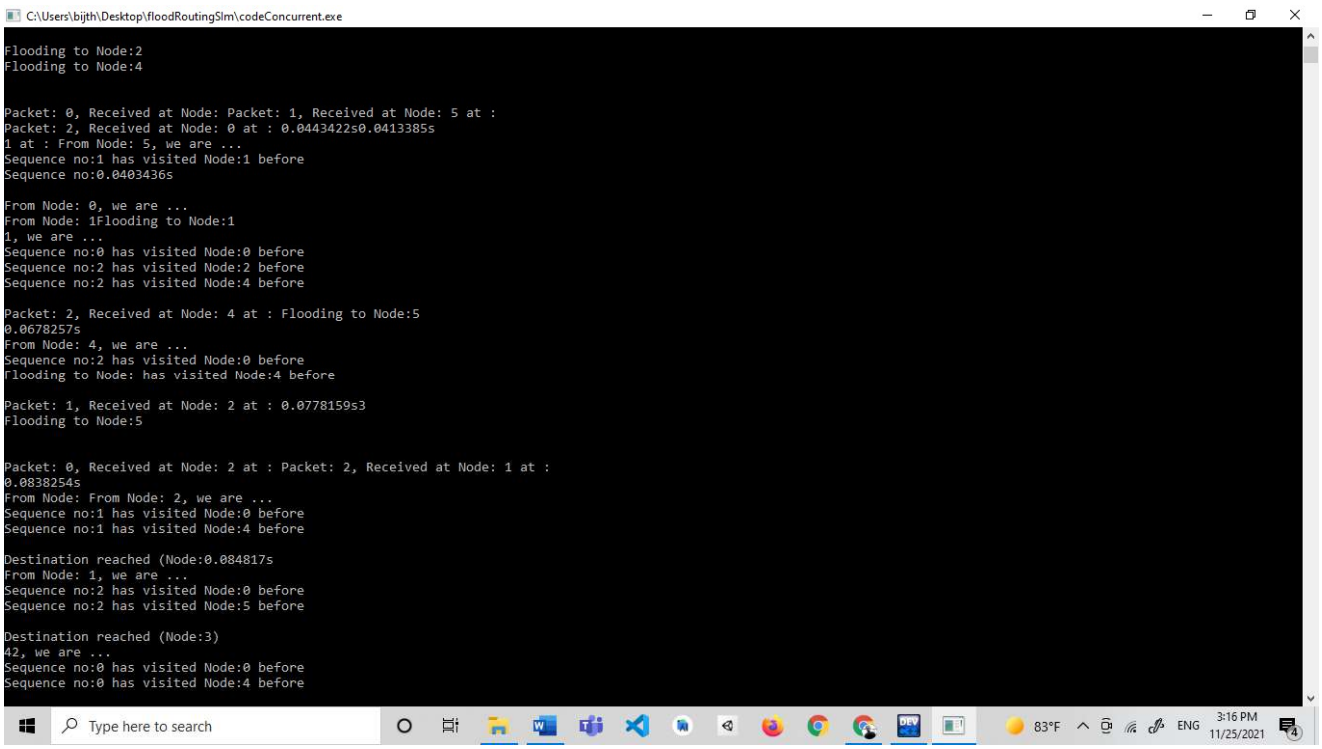

C:\Users\bijth\Desktop\floodRoutingSlm\codeNonConcurrent.exe
Sequence no:1 has visited Node:1 before
Flooding to Node:2
Flooding to Node:4
Packet: 1, Received at Node: 5 at : 0.0514627s
From Node: 5, we are ...
Sequence no:1 has visited Node:1 before
Sequence no:1 has visited Node:4 before
Packet: 1, Received at Node: 2 at : 0.0604624s
From Node: 2, we are ...
Sequence no:1 has visited Node:0 before
Sequence no:1 has visited Node:4 before
Destination reached (Node:4)
Packet:1 has been successfully received!
Sending Packet:2 at: 0.0764736s
Packet: 2, Received at Node: 2 at : 0.0794651s
From Node: 2, we are ...
flooding to Node:0
Flooding to Node:4
Packet: 2, Received at Node: 0 at : 0.0874631s
From Node: 0, we are ...
Flooding to Node:1
Sequence no:2 has visited Node:2 before
Sequence no:2 has visited Node:4 before
Packet: 2, Received at Node: 4 at : 0.0984622s
From Node: 4, we are ...
Sequence no:2 has visited Node:0 before
Flooding to Node:3
Flooding to Node:5
Sequence no:2 has visited Node:2 before
Packet: 2, Received at Node: 1 at : 0.112461s
From Node: 1, we are ...
Sequence no:2 has visited Node:0 before
Sequence no:2 has visited Node:5 before
Destination reached (Node:3)
Type here to search
83°F
ENG
3:15 PM
11/25/2021

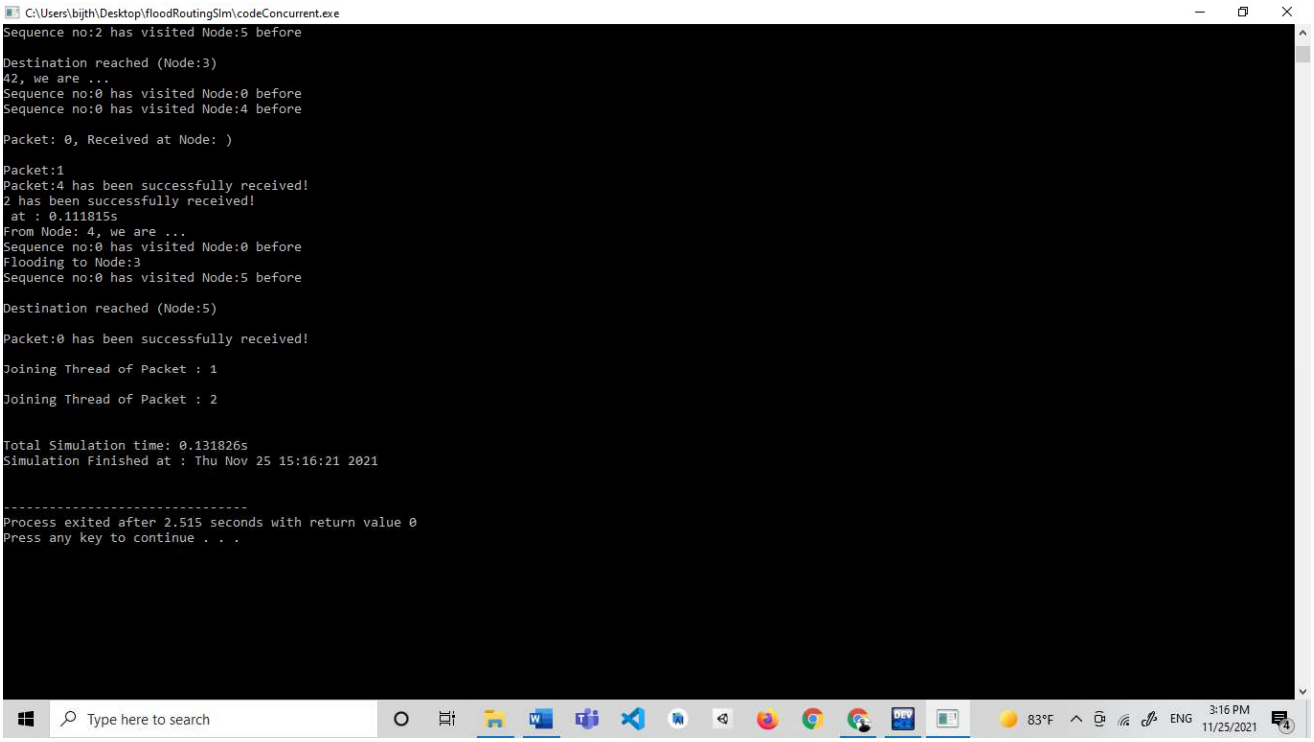

C:\Users\bijth\Desktop\floodRoutingSlm\codeNonConcurrent.exe
Sending Packet:2 at: 0.0764736s
Packet: 2, Received at Node: 2 at : 0.0794651s
From Node: 2, we are ...
Flooding to Node:0
Flooding to Node:4
Packet: 2, Received at Node: 0 at : 0.0874631s
From Node: 0, we are ...
Flooding to Node:1
Sequence no:2 has visited Node:2 before
Sequence no:2 has visited Node:4 before
Packet: 2, Received at Node: 4 at : 0.0984622s
From Node: 4, we are ...
Sequence no:2 has visited Node:0 before
Flooding to Node:3
Flooding to Node:5
Sequence no:2 has visited Node:2 before
Packet: 2, Received at Node: 1 at : 0.112461s
From Node: 1, we are ...
Sequence no:2 has visited Node:0 before
Sequence no:2 has visited Node:5 before
Destination reached (Node:3)
Packet:2 has been successfully received!
Total Simulation time: 0.124987s
Simulation Finished at : Thu Nov 25 15:14:55 2021

Process exited after 51.44 seconds with return value 0
Press any key to continue . . .
Type here to search
83°F
ENG
3:15 PM
11/25/2021

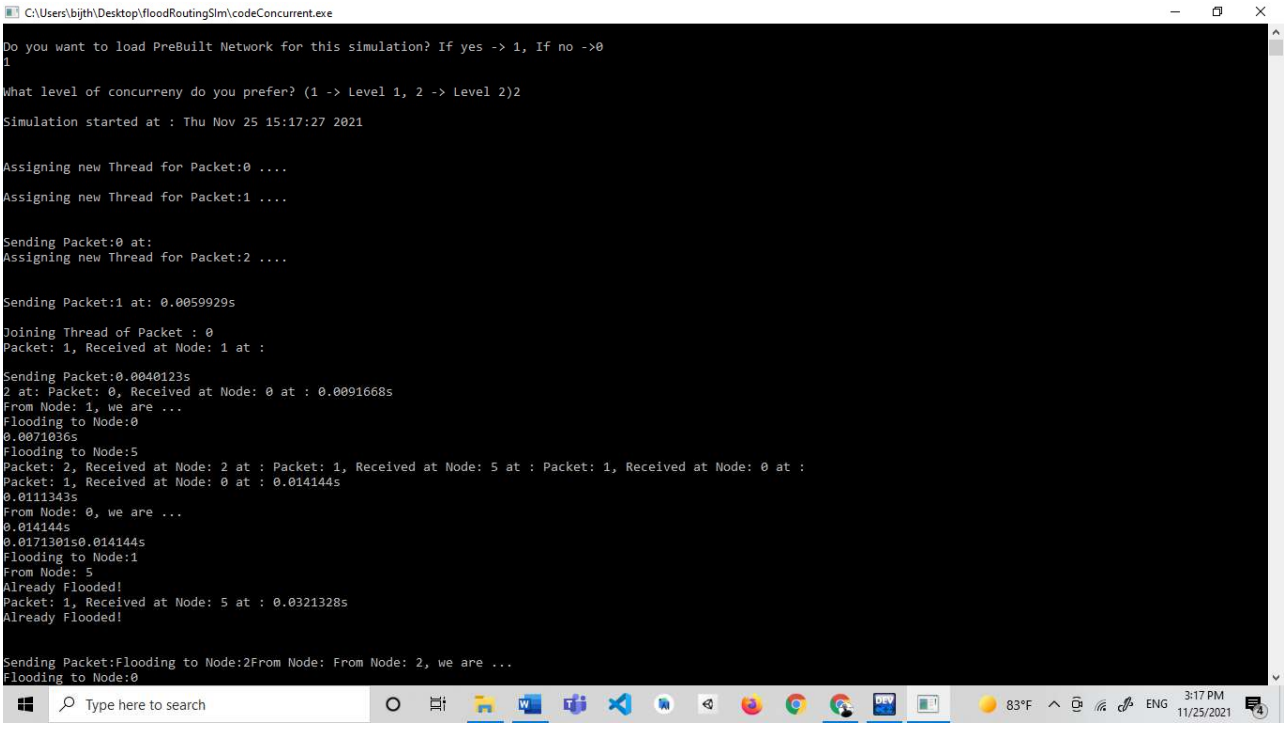
Concurrency level1:-

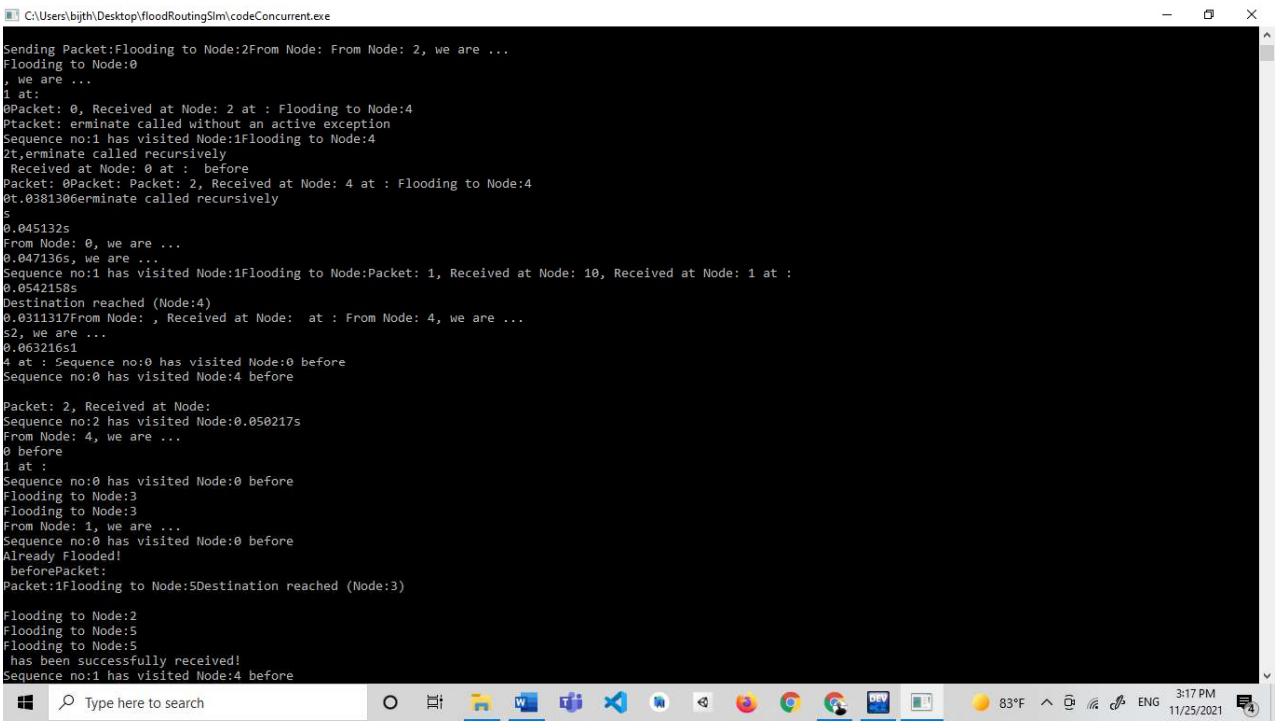


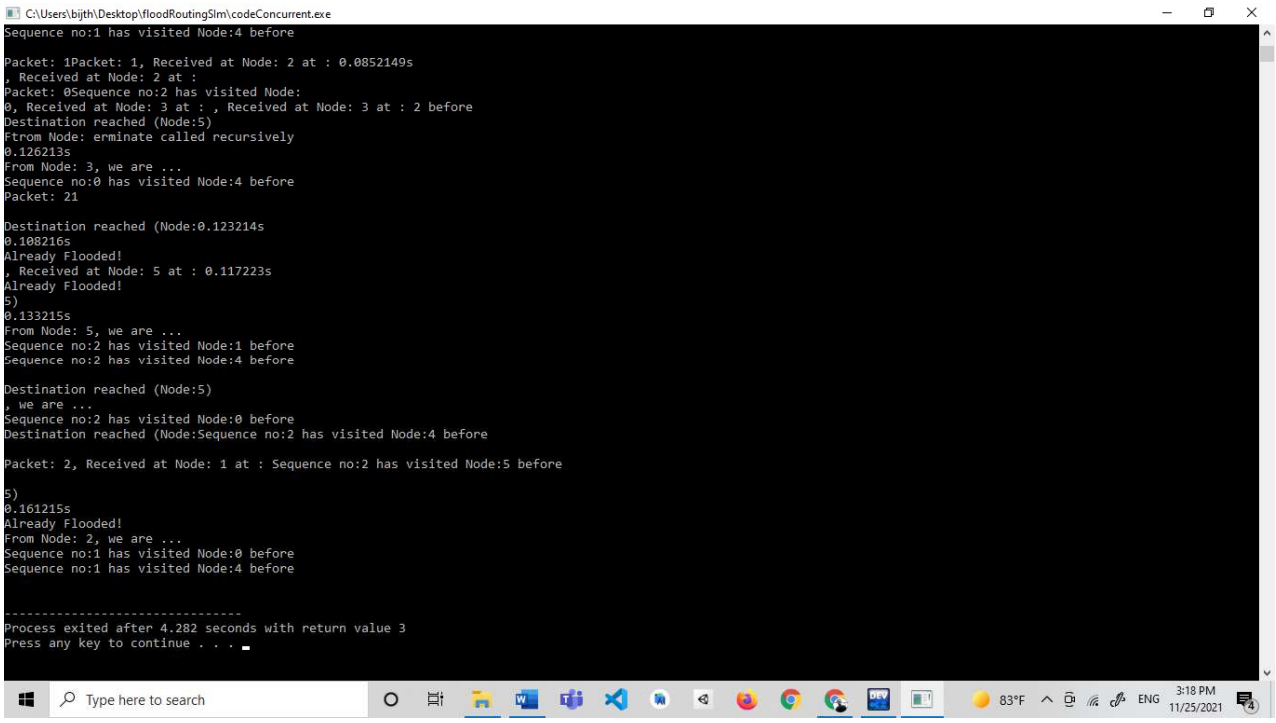




Concurrency Level2:-




C:\Users\bijth\Desktop\floodRoutingSlm\codeConcurrent.exe
Sending Packet:Flooding to Node:2From Node: From Node: 2, we are ...
Flooding to Node:0
, we are ...
1 at:
0Packet: 0, Received at Node: 2 at : Flooding to Node:4
Ptacket: erminate called without an active exception
Sequence no:1 has visited Node:1Flooding to Node:4
2t,erminate called recursively
Received at Node: 0 at : before
Packet: 0Packet: Packet: 2, Received at Node: 4 at : Flooding to Node:4
0t.0381306erminate called recursively
s
0.045132s
From Node: 0, we are ...
0.047136s, we are ...
Sequence no:1 has visited Node:1Flooding to Node:Packet: 1, Received at Node: 10, Received at Node: 1 at :
0.0542158s
Destination reached (Node:4)
0.0311317From Node: , Received at Node: at : From Node: 4, we are ...
s2, we are ...
0.0632161s1
4 at : Sequence no:0 has visited Node:0 before
Sequence no:0 has visited Node:4 before
Packet: 2, Received at Node:
Sequence no:2 has visited Node:0.050217s
From Node: 4, we are ...
0 before
1 at :
Sequence no:0 has visited Node:0 before
Flooding to Node:3
Flooding to Node:3
From Node: 1, we are ...
Sequence no:0 has visited Node:0 before
Already Flooded!
beforePacket:
Packet:1Flooding to Node:5Destination reached (Node:3)
Flooding to Node:2
Flooding to Node:5
Flooding to Node:5
has been successfully received!
Sequence no:1 has visited Node:4 before
Type here to search
83°F
ENG
3:17 PM
11/25/2021


C:\Users\bijth\Desktop\floodRoutingSlm\codeConcurrent.exe
Sequence no:1 has visited Node:4 before
Packet: 1Packet: 1, Received at Node: 2 at : 0.0852149s
, Received at Node: 2 at :
Packet: 0Sequence no:2 has visited Node:
0, Received at Node: 3 at : , Received at Node: 3 at : 2 before
Destination reached (Node:5)
Ftrom Node: erminate called recursively
0.126213s
From Node: 3, we are ...
Sequence no:0 has visited Node:4 before
Packet: 21
Destination reached (Node:0.123214s
0.108216s
Already Flooded!
, Received at Node: 5 at : 0.117223s
Already Flooded!
5)
0.133215s
From Node: 5, we are ...
Sequence no:2 has visited Node:1 before
Sequence no:2 has visited Node:4 before
Destination reached (Node:5)
, we are ...
Sequence no:2 has visited Node:0 before
Destination reached (Node:Sequence no:2 has visited Node:4 before
Packet: 2, Received at Node: 1 at : Sequence no:2 has visited Node:5 before
5)
0.161215s
Already Flooded!
From Node: 2, we are ...
Sequence no:1 has visited Node:0 before
Sequence no:1 has visited Node:4 before
Process exited after 4.282 seconds with return value 3
Press any key to continue . . .
Type here to search
83°F
ENG
3:18 PM
11/25/2021

Solutions I used to overcome looping:-

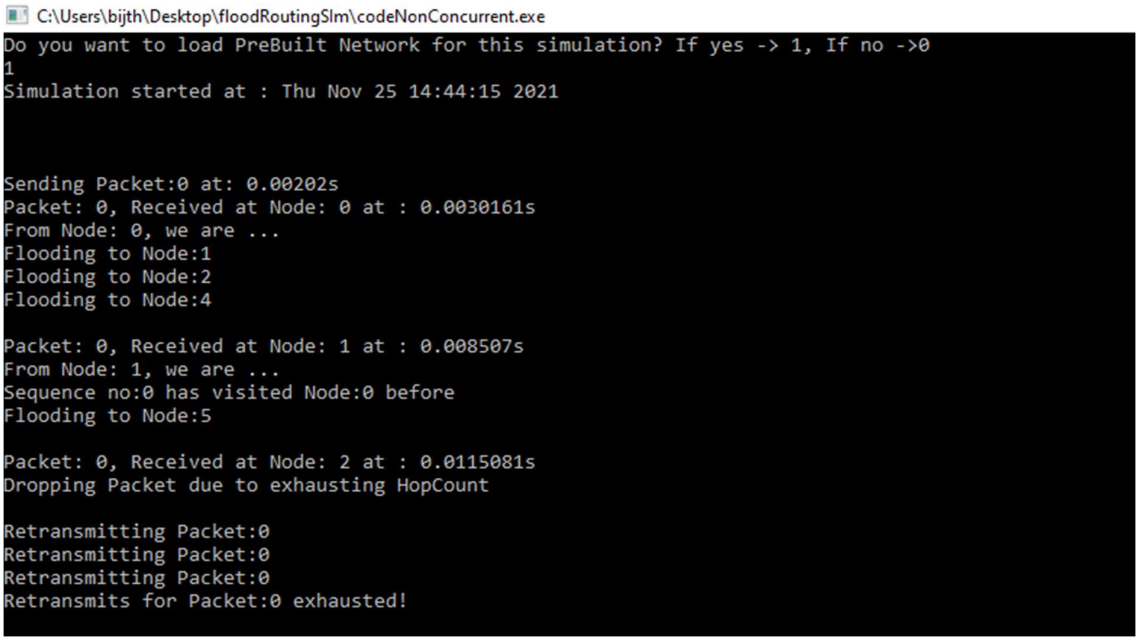
Showing Outputs based on

1. MaxHopCount:-

As seen before, each packet has a maxHopCount that is set before simulation begins. Once the hopCount hits zero, the packet is discarded.

Pre-set the maxHopCount to be 3.

nonConcurrent:-



concurrent lvl1:-

(Do Scroll Down)

```
Packet: 0, Received at Node: 1 at : Packet: 1, Received at Node: 1 at : 0.0095105s
0.01551260.0105124s
From Node: 1, we are ...
Flooding to Node:0
Flooding to Node:5

Packet: 1, Received at Node: 0 at : s
From Node: 10.0225108s
Dropping Packet due to exceeding TTL
, we are ...
Sequence no:0 has visited Node:0 before
Flooding to Node:5
From Node: 2, we are ...
Flooding to Node:0

Packet: 0, Received at Node: 2 at : 0.0355122s
Dropping Packet due to exceeding TTL
Flooding to Node:4

Packet: 2, Received at Node: 0 at : 0.0434061s
Dropping Packet due to exceeding TTL

Retransmitting Packet:1
Retransmitting Packet:2
Retransmitting Packet:0
Retransmitting Packet:
Retransmitting Packet:12
Retransmitting Packet:0
Retransmitting Packet:1
Retransmitting Packet:2
Retransmitting Packet:0
Retransmits for Packet:2 exhausted!

Retransmits for Packet:1 exhausted!

Retransmits for Packet:0 exhausted!
```

concurrent lvl2:-

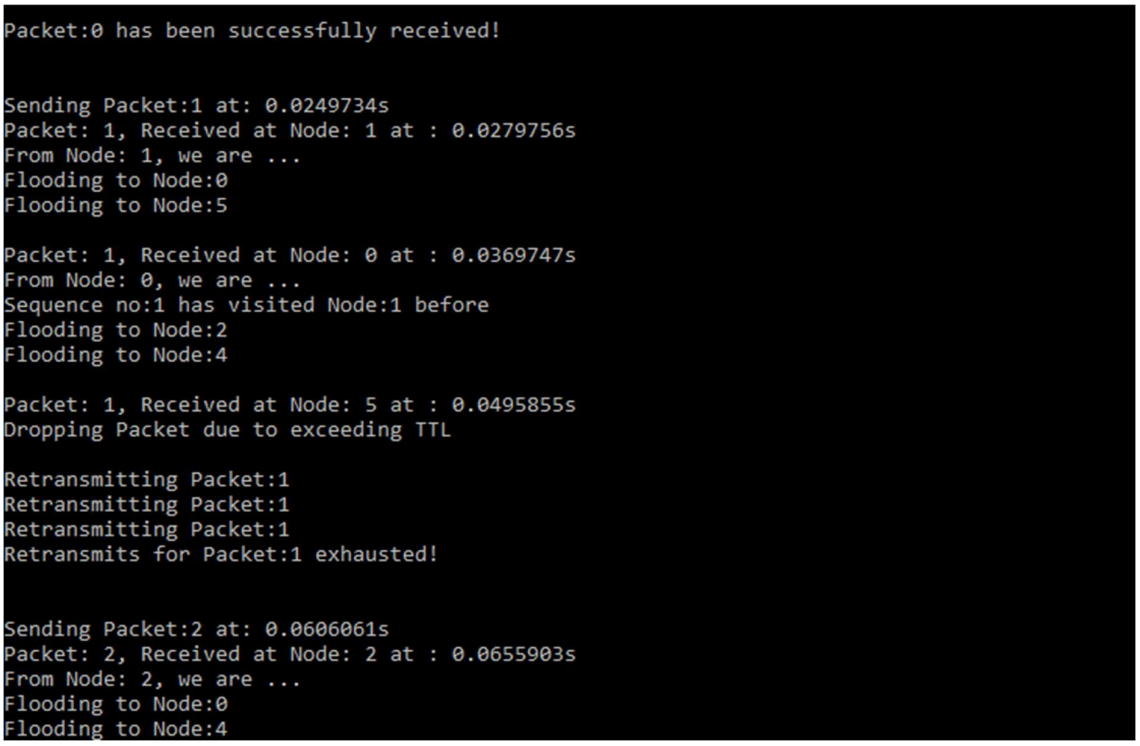
```
Destination reached (Node:0.0199143Packet: , Received at Node: 2s
Flooding to Node:5Destination reached (Node:3)
0 at : , Received at Node: 0 at : 0.0489605s
Already Flooded!
Flooding to Node:5
Ptacket: erminate called recursively
5)
Dropping Packet due to exceeding TTL
From Node: 2, we are ...
Sequence no:0 has visited Node:0 beforeDropping Packet due to exceeding TTL

Packet: 2, Received at Node: 5 at :
Packet: 1, Received at Node: 0 at : 0.0709637s
Dropping Packet due to exceeding TTL
0.0509611
Sequence no:0 has visited Node:4 before0.0819615s
Dropping Packet due to exceeding TTL
s
Packet: 2, Received at Node:
Dropping Packet due to exceeding TTL
1Sequence no:1, Received at Node: 5 at : 20.0809615
 at : 0.0639611s
Dropping Packet due to exceeding TTL
 has visited Node:s
2 before
Sequence no:Dropping Packet due to exceeding TTL
2 has visited Node:4 before

Packet: 2, Received at Node: 4 at : 0.107962s
Dropping Packet due to exceeding TTL

Packet:2 has been successfully received!
```

2. Total Time to Live:-

The packet is assigned a TTL and is timestamped for its start point. Based on its start point and current time, if it has exceeded TTL, it is discarded

Pre-set the TTL_left to be 0.02s.

nonConcurrent:-



(Do Scroll Down)

concurrent lvl1:-

```
Packet: 0, Received at Node: 1 at : 0.0096143s
From Node: 1, we are ...
Sequence no:0 has visited Node:0 before
Flooding to Node:5
0.0040057s

Packet: 0, Received at Node: 2 at : Packet: 1, Received at Node: 1 at :

Sending Packet:2 at: 0.0186136s
From Node: 20.0176129s
0.0050016s
From Node: 1, we are ...
Flooding to Node:0
Flooding to Node:5

Packet: , we are ...
Packet: 1, Received at Node: 0 at : 2, Received at Node: 2 at : 0.0286139Sequence no:0 has visited Node:0.0346292s
Dropping Packet due to exceeding TTL
0 before
Sequence no:0 has visited Node:4 before

Packet: 0, Received at Node: 4 at : s
Dropping Packet due to exceeding TTL
0.0485274s
Dropping Packet due to exceeding TTL

Retransmitting Packet:1
Retransmitting Packet:2
Retransmitting Packet:0
Retransmitting Packet:1
Retransmitting Packet:0
Retransmitting Packet:2
Retransmitting Packet:1
Retransmitting Packet:
Retransmitting Packet:02
Retransmits for Packet:1 exhausted!

Retransmits for Packet:
Retransmits for Packet:2 exhausted!
```

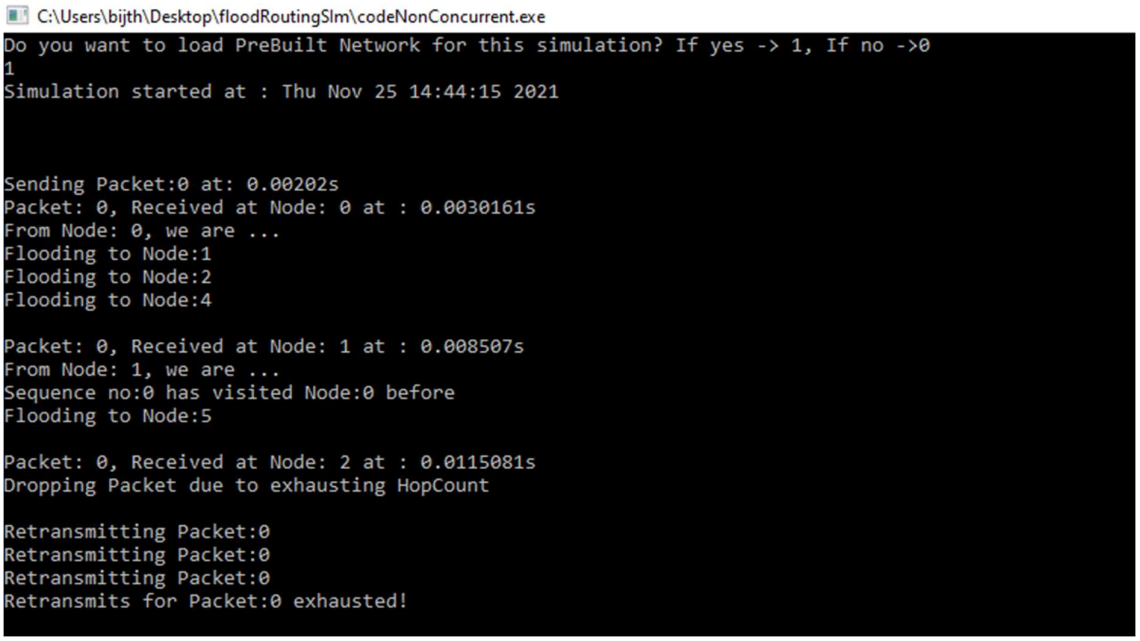
concurrent lvl2:-

```
Sending Packet:2 at: , Received at Node: 0 at : 0.0080029sPacket: 1, Received at Node: 1 at :
Joining Thread of Packet : 0
0.0070017s

From Node: 0, we are ...
Flooding to Node:1
0.0080029s
Packet: 2, Received at Node: 2 at : Packet: 0, Received at Node: 0.0167008s
1 at : From Node: 1, we are ...
Flooding to Node:0
0.0167008s
Flooding to Node:5
Flooding to Node:2
From Node: 2, we are ...
Flooding to Node:0
Packet: 0, Received at Node: 2 at : Flooding to Node:4

tPerminate called without an active exception
acket: 1, Received at Node: 0.0276987s
Dropping Packet due to exceeding TTL
Packet: From Node: 1, we are ...
Flooding to Node:4
0, Received at Node: Packet: 2, Received at Node: 0 at :
Packet: 2, Received at Node: 0 at : Packet: 2, Received at Node: 4 at : 5 at : 0.0437036s
Dropping Packet due to exceeding TTL
Sequence no:0 has visited Node:0 before
Flooding to Node:5
0.0267003sPacket: Destination reached (Node:
0.0306991s
0.0427158s
Dropping Packet due to exceeding TTL
Destination reached (Node:5)
```

3. Controlled Retransmissions:-

Packets are retransmitted only after an acknowledge that it has been dropped is received. Also, for the concurrent mode, it retransmits only after a sec on receiving drop ack.

nonConcurrent:-



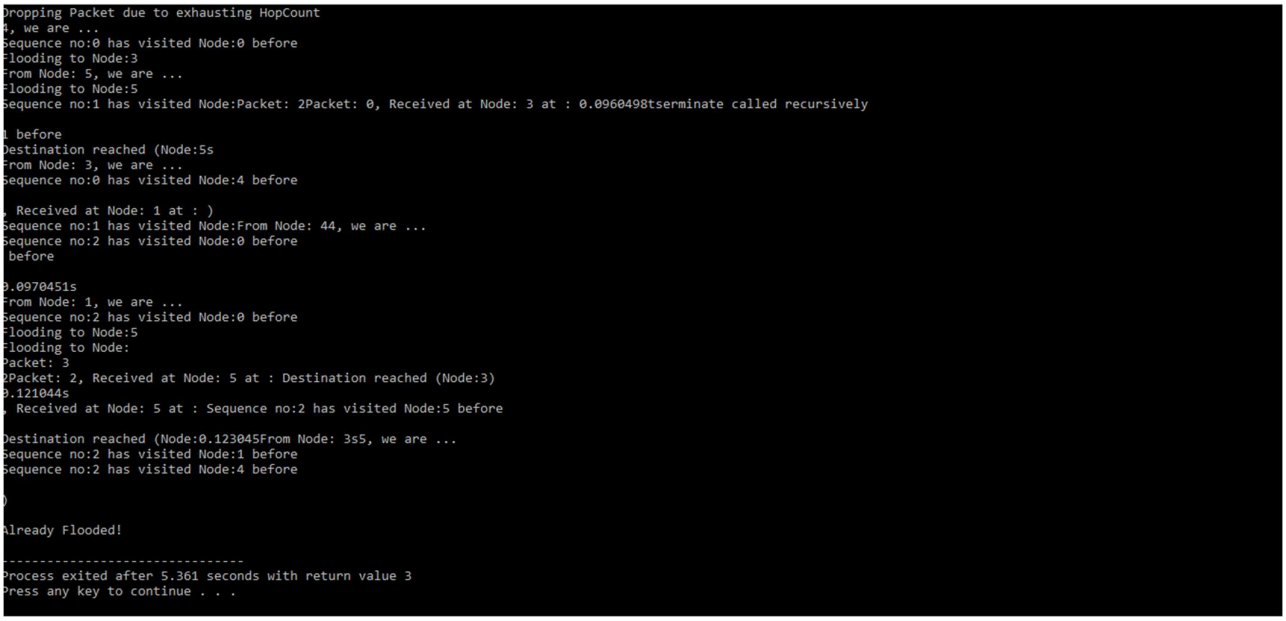
(Do Scroll Down)

concurrent lvl1:-

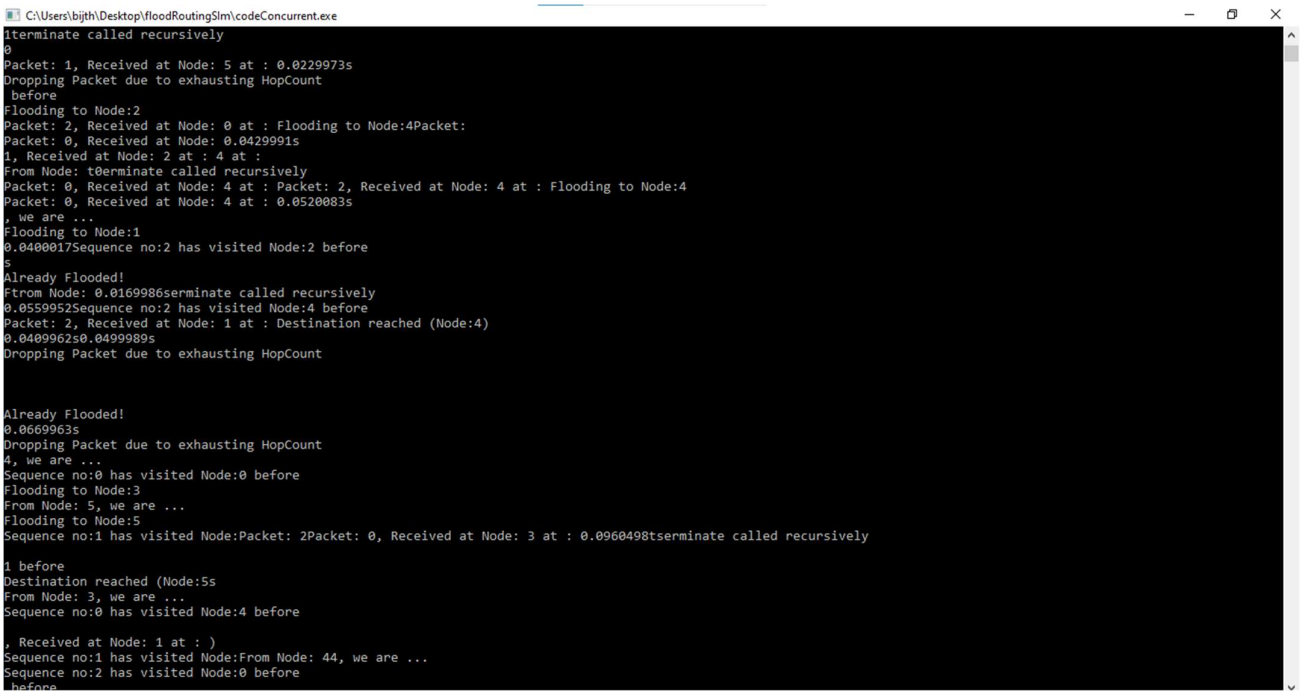
```
Packet: 1, Received at Node: 5 at : From Node: 2, we are ...
Flooding to Node:20.0300671s
Dropping Packet due to exhausting HopCount

Flooding to Node:0
Flooding to Node:4

Packet: 0, Received at Node: 1 at : Flooding to Node:4

Packet: 2, Received at Node: 0 at : 0.0440012s
0.0470079s
From Node: 0, we are ...
Flooding to Node:1From Node: 1, we are ...

Sequence no:2 has visited Node:2 before
Sequence no:2 has visited Node:4 before

Sequence no:0 has visited Node:0 before
Flooding to Node:5

Packet: 0, Received at Node: 2 at : Packet: 2, Received at Node: 4 at : 0.0693279s
Dropping Packet due to exhausting HopCount
0.0663187s
Dropping Packet due to exhausting HopCount

Retransmitting Packet:1
Retransmitting Packet:2
Retransmitting Packet:0
Retransmitting Packet:1
Retransmitting Packet:0
Retransmitting Packet:2
Retransmitting Packet:1
Retransmitting Packet:0
Retransmitting Packet:2
Retransmits for Packet:1 exhausted!

Retransmits for Packet:2 exhausted!

Retransmits for Packet:0 exhausted!

Joining Thread of Packet : 1
```

concurrent lvl2:-

```
C:\Users\bijth\Desktop\floodRoutingSlm\codeConcurrent.exe

Do you want to load PreBuilt Network for this simulation? If yes -> 1, If no ->0
1

What level of concurreny do you prefer? (1 -> Level 1, 2 -> Level 2)2

Simulation started at : Thu Nov 25 15:06:25 2021


Assigning new Thread for Packet:0 ....

Assigning new Thread for Packet:1 ....


Sending Packet:0 at:
Assigning new Thread for Packet:2 ....


Sending Packet:1 at:

Sending Packet:2 at: 0.0029962s

Joining Thread of Packet : 0
Packet: 1, Received at Node: 1 at : 0.0080112s
From Node: 1, we are ...
Flooding to Node:0
0.0029962s
0.0059992s
Packet: Packet: Flooding to Node:Packet: 1, Received at Node: 0 at : 0, Received at Node: 0 at : 5
0.0109948s2t,erminate called without an active exception
 Received at Node: 2 at : 0.0109948s
0.0109948s
From Node: 0, we are ...
Flooding to Node:1
From Node: 2, we are ...
Flooding to Node:2

From Node: 0, we are ...
Sequence no:1Packet: 0Flooding to Node:4
 has visited Node:Flooding to Node:, Received at Node: 2 at : 0.0279995s
Packet: 0, Received at Node: 1 at : From Node: 2, we are ...
Sequence no:0 has visited Node:0 before
Flooding to Node:4
1terminate called recursively
```



```
Dropping Packet due to exhausting HopCount
4, we are ...
Sequence no:0 has visited Node:0 before
Flooding to Node:3
From Node: 5, we are ...
Flooding to Node:5
Sequence no:1 has visited Node:Packet: 2Packet: 0, Received at Node: 3 at : 0.0960498tserminate called recursively

1 before
Destination reached (Node:5s
From Node: 3, we are ...
Sequence no:0 has visited Node:4 before

, Received at Node: 1 at : )
Sequence no:1 has visited Node:From Node: 44, we are ...
Sequence no:2 has visited Node:0 before
 before

0.0970451s
From Node: 1, we are ...
Sequence no:2 has visited Node:0 before
Flooding to Node:5
Flooding to Node:
Packet: 3
2Packet: 2, Received at Node: 5 at : Destination reached (Node:3)
0.121044s
, Received at Node: 5 at : Sequence no:2 has visited Node:5 before

Destination reached (Node:0.123045From Node: 3s5, we are ...
Sequence no:2 has visited Node:1 before
Sequence no:2 has visited Node:4 before

)

Already Flooded!

--------------------------------
Process exited after 5.361 seconds with return value 3
Press any key to continue . . .
```



```
C:\Users\bijth\Desktop\floodRoutingSlm\codeConcurrent.exe
1terminate called recursively
0
Packet: 1, Received at Node: 5 at : 0.0229973s
Dropping Packet due to exhausting HopCount
 before
Flooding to Node:2
Packet: 2, Received at Node: 0 at : Flooding to Node:4Packet:
Packet: 0, Received at Node: 0.0429991s
1, Received at Node: 2 at : 4 at :
From Node: t0erminate called recursively
Packet: 0, Received at Node: 4 at : Packet: 2, Received at Node: 4 at : Flooding to Node:4
Packet: 0, Received at Node: 4 at : 0.0520083s
, we are ...
Flooding to Node:1
0.0400017Sequence no:2 has visited Node:2 before
s
Already Flooded!
Ftrom Node: 0.0169986serminate called recursively
0.0559952Sequence no:2 has visited Node:4 before
Packet: 2, Received at Node: 1 at : Destination reached (Node:4)
0.0409962s0.0499989s
Dropping Packet due to exhausting HopCount



Already Flooded!
0.0669963s
Dropping Packet due to exhausting HopCount
4, we are ...
Sequence no:0 has visited Node:0 before
Flooding to Node:3
From Node: 5, we are ...
Flooding to Node:5
Sequence no:1 has visited Node:Packet: 2Packet: 0, Received at Node: 3 at : 0.0960498tserminate called recursively

1 before
Destination reached (Node:5s
From Node: 3, we are ...
Sequence no:0 has visited Node:4 before

, Received at Node: 1 at : )
Sequence no:1 has visited Node:From Node: 44, we are ...
Sequence no:2 has visited Node:0 before
 before
```

(Do Scroll Down)

4. Keeping Track of Flooded Packets & Nodes Visited:-

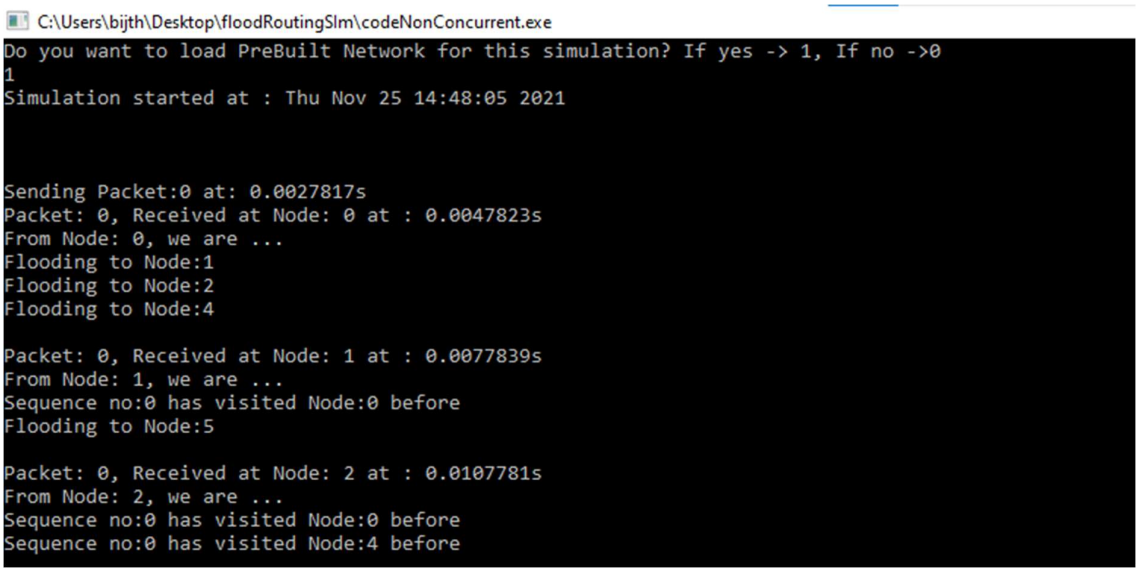
I. unordered<set> nodesVisited:-

This is with respect to packet. It keeps track of all the nodes it has visited. Thus, when it is on current node, the node can use this info from nodesVisited to avoid sending a particular packet to an already visited node. It works in sync with floodedPackets.

II. unordered<int> floodedPackets:-

floodedPackets keeps track of the packets a current node has flooded already. This helps avoiding the same node to sent a previously flooded packet again (Comes into Play powerfully when level of concurrency is 2)

nonConcurrent:-

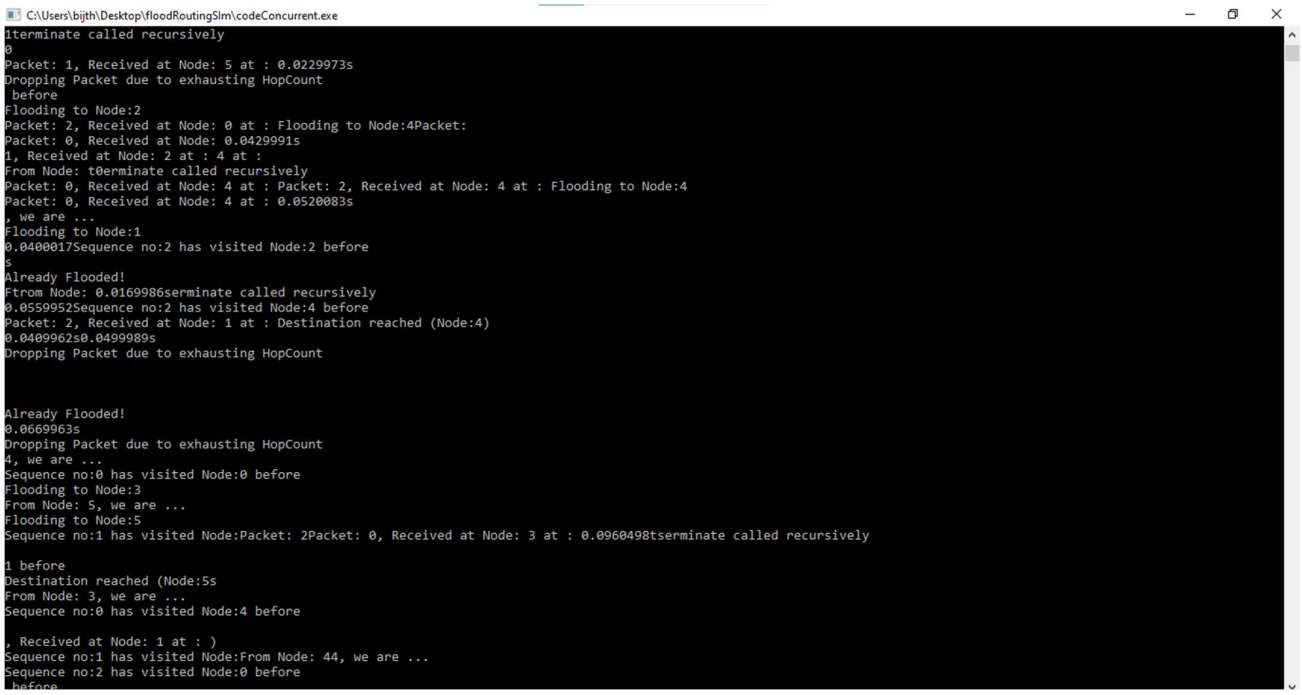


(Do Scroll Down)

concurrent lvl1:-

```
Packet: 0, Received at Node: 1 at : 0.0090028s
From Node: 2, we are ...
Flooding to Node:0
Flooding to Node:40.0080356s
0.0160078s
From Node: 1, we are ...
Sequence no:0 has visited Node:0 before
Flooding to Node:5

Packet:
From Node: 1, we are ...
Flooding to Node:0

0, Received at Node: 2 at : Packet: 2, Received at Node: 00.031924Flooding to Node:5s
 at :

Packet: 1, Received at Node: 0 at : Dropping Packet due to exhausting HopCount
0.0379267s
From Node: 0, we are ...
Flooding to Node:1
Sequence no:0.04392482 has visited Node:2 before
Sequence no:2 has visited Node:4 before
```

concurrent lvl2:-



THANK YOU, MA'AM!