I.T.U.

Faculty of Computer and Informatics

Computer Engineering

Lesson name: Computer Communications

Lesson Code: BLG 433E

Instructor's Name: Sema OKTUĞ

Name Surname: Abdullah AYDEĞER

Number: 040090533

Due Date: 30.12.2012

Table of Contents

Introduction

In computer network area routing is very important since main function of network layer of communication is routing packages from source to destination. Routing algorithms are used for this routing operation. In this project, Link State Routing method is implemented for routing the packages.

Link State Routing

A Link-state routing is a concept used in routing of packet-switched networks in computer communications. Link-state routing works by having the routers tell every router on the network about its closest neighbors. The entire routing table is not distributed from any router, only the part of the table containing its neighbors.

Link state routing has some variants named as IS-IS and OSPF that are widely used in networks.

Link state routing can be divided into 5 substances:

1. Discover its neighbors and learn their network addresses.
2. Set the distance or cost metric to each of its neighbors.
3. Construct a packet telling all it has just learned.
4. Send this packet to and receive packets from all other routers.
5. Compute the shortest path to every other router.

In my project's source code, first 3 steps were implemented in createAllPackets function as seen below:

```
vector<Packet> Packet::createAllPackets(Node *n, int age, int numOfNode){
    vector<Packet> vPack;
    Link l;
    list<Link>::iterator it;
    for(int i=0; i<numOfNode; i++){ //for all nodes in network
        Packet p;
        p.age = age;
        p.timeOfPath = 0;   //for creating, initialize cost as '0'
        for(it = n[i].connectedNodes.begin(); it != n[i].connectedNodes.end(); it++){   //for all neighbours
            l = (*it);
            p.packetData.push_back(l);  //learn about it
    |   }
        p.nodesToSent = p.packetData;   //firstly nodesToSent will equal to packetData since node will sent packet to nodes which are already in packet
        p.ownerId = i;  //owner of packet initialized
        p.currentNode = i;
        p.parentId = -1;    //it does not have any parent
        vPack.push_back(p);
    }
    return vPack;   //return back to all packages which are created for nodes
}
```

Figure 1. createAllPackets

4th and 5th steps were implemented at packetBroading function that distribute packages according to delay of links as seen below:

```
int Packet::packetBroading(Node *n, int age, int numOfNode){
    Link l;
    Packet p;
    Pair pair, pair2;
    list<Link>::iterator it;
    Packet oldPacket;
    int order, broadCastTime = 0;

    for(int cl=0; cl<numOfNode; cl++)
        n[cl].clearRouting(numOfNode);

    vector<Packet> allPacketsToSent = createAllPackets(n, age, numOfNode);

    while(!allPacketsToSent.empty()){       //wait for all packets to die
        oldPacket = allPacketsToSent[0];
        p.ownerId = oldPacket.ownerId;
        p.parentId = oldPacket.currentNode;
        p.age = oldPacket.age -1;
        allPacketsToSent.erase(allPacketsToSent.begin());
        if(oldPacket.age > 0){  //if it is not invalid(not died)
       for(int i = 0; i < oldPacket.nodesToSent.size(); i++){     //Sent packet to all neighbours
           order = oldPacket.nodesToSent[i].numberToLink;
           pair.num1 = oldPacket.ownerId;
           pair.num2 = oldPacket.nodesToSent[i].cost + oldPacket.timeOfPath;
           pair2 = n[order].routingTable[oldPacket.ownerId];
           p.currentNode = order;
           if(oldPacket.ownerId == oldPacket.currentNode){ //for first packages
               if(pair2.num2 > pair.num2){
                   n[order].routingTable[oldPacket.ownerId] = pair;    //update routing-table
               }
           }
           else if(pair2.num2 > pair.num2 && oldPacket.ownerId != order){
               pair.num1 = oldPacket.currentNode;
               n[order].routingTable[oldPacket.ownerId] = pair;
           }
           //new packages will be created at this below if
           if(n[p.ownerId].isAdded[p.currentNode] == -1){
               p.packetData = oldPacket.packetData;
               p.timeOfPath = oldPacket.timeOfPath;
               p.timeOfPath += oldPacket.nodesToSent[i].cost;
               for(it = n[order].connectedNodes.begin(); it != n[order].connectedNodes.end(); it++){
                   l = (*it);
                   p.nodesToSent.push_back(l);
               }
               p.addPacketToAll(&allPacketsToSent);
               n[p.ownerId].isAdded[p.currentNode] = 1;
               p.nodesToSent.clear();
           }
       }
    }
```

Figure 2. packetBroading

Simulation Parameters

By giving simulation parameters as constant at the beginning of the main.c file, we can easily simulate our network with different characteristics such as node number, failure times.

Number of Bidirectional Links

In this project it is given as 20 and for trying to other network topologies it can be changed through changing constant NUMBEROFLINKS parameter in main.c file.

```
#define NUMBEROFLINKS 20
```

Figure 3. NUMBEROFLINKS

Number of Nodes in Network

In this project it is given as 8. But this is not strict since its value can be increased according to random number between 0 and RANDOMNODE(will be explained). That means our total node number will be between 8-12. We can change this number of nodes in the network by changing constant MINUMUMNODE parameter in main.c file.

```
#define MINUMUMNODE 8
```

Figure 4. MINUMUMNODE

```
int randNode = rand() % RANDOMNODE;
randNode += MINUMUMNODE;
```

Figure 5. Total number of nodes in network

Number of Random Nodes in Network

In this project it is given as 4. Because between 8 and 12 nodes are required to generate, then random nodes should not exceed 4. We can change this random number of nodes in the network by changing constant RANDOMNODE parameter in main.c file.

```
#define RANDOMNODE 4
```

Figure 6. RANDOMNODE

Time of Simulation

In this project network routings will be simulated as 40 hours and it is determined as given constant MAXHOURS parameters. We can change this time of simulation by changing constant MAXHOURS parameter in main.c file.

```
#define MAXHOURS 40
```

Figure 7. MAXHOURS

Cost of Link between two Nodes

In this Project cost of each links has an upper bound as 10 and a lower bound as 1. This cost in type of miliseconds and is not important delay fort his Project. But even for control of total routing cost if it exceeds 10 seconds, then time count for 10 seconds is increased. We can change this number of nodes in the network by changing constant MAXCOST parameter in Node.h file.

```
#define MAXCOST 10
```

Figure 8. MAXCOST

```
broadCastTime = pac.packetBroading(n, randNode+1-((NUMBEROFLINKS-randNode)/2) , randNode);      //for each 10sec.
timeCount = timeCount + broadCastTime/10000;    //broadcasting of all packages can exceed 10sec for later usages
```

Figure 9. Control of broad casting time

Routing Cost of any two Nodes

Because of the fact that link state routing is used in this Project, routing cost should be initialized as infinity at the first of broad casting. For using infinity in the Project, MAXROUTE as defined COST*9999 and if more than 9999 nodes are given for simulation, the simulation may fail.

```
#define MAXROUTE MAXCOST*9999   //this will adjust routing of all nodes to 99999(like infinity)
```

Figure 10. MAXROUTE

Mean Value of Failure of Links

In this Project failure of links is required to have 2 average hour. Then this value is defined as constant and can be changed by replacing new value into the FAILUREMEAN in main.c file.

```
#define FAILUREMEAN 2
```

Figure 11. FAILUREMEAN

Mean Value of Recovery of Failed Links

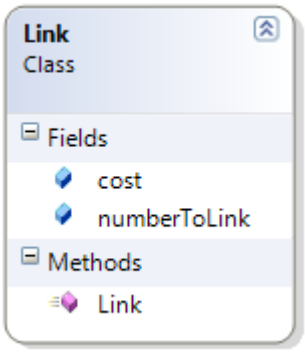
In this Project recovery of failed links is required to have 3 average hour. Then this value is defined as constant and can be changed by replacing new value into the RECOVERYMEAN in main.c file.

```
#define RECOVERYMEAN 3
```

Figure 12. RECOVERYMEAN

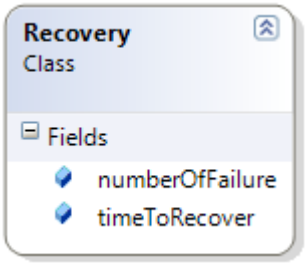
Written Classes for Simulation

Link



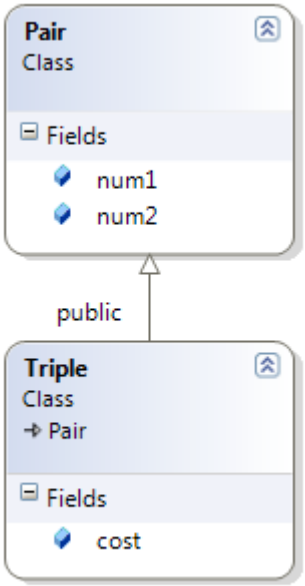
This class is almost same as Pair class. But the purpose of written of this class is for next usage of Project some other features can be added to links directly by using this class.

Recovery



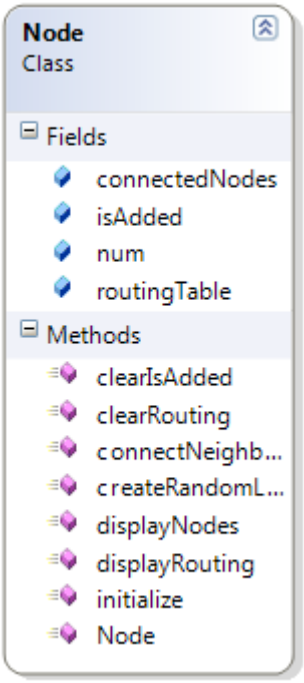
This class is keeping recover of each failures' time.

Pair & Triple



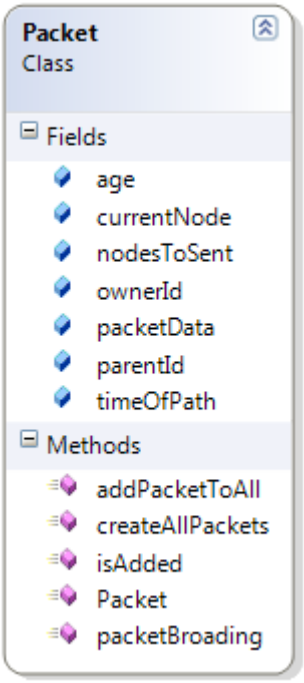
These classes will be used for keeping two or three integer in one variable.

Node



This is main node class which represent nodes in network.

Packet



This class is main package class that keeps necessary information for package to sent.

Simulation Compiling and Running

Compiling

Project source code is written in Win7 Microsoft Visual Studio 2010 and compiled at both Visual Studio, Ubuntu 11.04 and ITU's Linux Server without any error. At ITU's Linux Server for compiling the Project code it is required to run make command in the source code folder.

Running

For running Project at ITU just run the ./comcom command with valid parameters in main.c.

Figure 13. Running the project

If parameters are not valid, the program will directly exit as seen below.

```
if(randNode+1> NUMBEROFLINKS || (randNode*(randNode-1))/2 < NUMBEROFLINKS){     //Invalid parameters
    cerr << "Number of links: " << NUMBEROFLINKS << " and number of nodes: " << randNode << endl;
    cerr << "With these parameters program can not generate connected graph\n";
    getchar();
    exit(0);
}
```

Figure 14. Invalid Parameters

One example running screen of program from SSH:

```
Links of node5:
        Node num:1 Cost: 7
        Node num:2 Cost: 2
        Node num:4 Cost: 3
        Node num:3 Cost: 7
        Node num:7 Cost: 1
        Node num:6 Cost: 2


Links of node6:
        Node num:0 Cost: 2
        Node num:7 Cost: 3
        Node num:2 Cost: 6
        Node num:5 Cost: 2
        Node num:1 Cost: 10


Links of node7:
        Node num:6 Cost: 3
        Node num:3 Cost: 5
        Node num:2 Cost: 10
        Node num:1 Cost: 2
        Node num:5 Cost: 1
        Node num:0 Cost: 10


 Please select route node number in all nodes
```

Figure 15. Example screen while running

Next steps while running as following:

```
 Please select route node number in all nodes 7
Routing table of selected node:
        For node: 0: 1 with cost: 3
        For node: 1: 1 with cost: 2
        For node: 2: 5 with cost: 3
        For node: 3: 3 with cost: 5
        For node: 4: 5 with cost: 4
        For node: 5: 5 with cost: 1
        For node: 6: 6 with cost: 3
        For node: 7: 7 with cost: 0

-----Failure link is between 6 and 0 at time 1 hour 3 minute 0 seconds-----

*****After link failure or recovery*****
Routing table of selected node:
        For node: 0: 1 with cost: 3
        For node: 1: 1 with cost: 2
        For node: 2: 5 with cost: 3
        For node: 3: 3 with cost: 5
        For node: 4: 5 with cost: 4
        For node: 5: 5 with cost: 1
        For node: 6: 6 with cost: 3
        For node: 7: 7 with cost: 0
For continuing please press the enter

-----Failure link is between 6 and 1 at time 1 hour 21 minute 20 seconds-----

*****After link failure or recovery*****
Routing table of selected node:
        For node: 0: 1 with cost: 3
        For node: 1: 1 with cost: 2
        For node: 2: 5 with cost: 3
        For node: 3: 3 with cost: 5
        For node: 4: 5 with cost: 4
        For node: 5: 5 with cost: 1
        For node: 6: 6 with cost: 3
        For node: 7: 7 with cost: 0
For continuing please press the enter
```

Figure 16. Example screen-2 while running

Simulation Results

Mean Time between Failures

Mean time for failures is calculated as following equation:

Mean time = (Last-failure-time) / (Number-of-failures)

Mean Time between Recovery of Failed Links

Mean time for recoveries of failed links is calculated as following equation:

Mean time = (Last-recovery-time) – (First-failure-time) / (Number-of-recoveries)

```
***Mean value of failures:1.09689

***Mean value of recoveries:1.18628
```

Figure 17. Mean values of recovery and fail times

Comments

Project Drawback

In this Project it is declared that print the updated routing table just after 1 minute from the failure or recovery. But as seen in the below figure, sometimes it is not possible to print routing table after failure or recovery. For a better solution; Instead of 1 minute waiting to print, 20 second waiting is more convenient to do that.

```
-----Failure link is between 6 and 5 at time 17 hour 34 minute 0 seconds-----

+++++Recovery link is between 4 and 1 at time 17 hour 34 minute 30 seconds+++++

*****After link failure or recovery*****
Routing table of selected node:
        For node: 0: 1 with cost: 3
        For node: 1: 1 with cost: 2
        For node: 2: 6 with cost: 9
        For node: 3: 3 with cost: 5
        For node: 4: 6 with cost: 6
        For node: 5: 1 with cost: 9
        For node: 6: 6 with cost: 3
        For node: 7: 7 with cost: 0
For continuing please press the enter
```

Figure 18. Project drawback

Link State Routing

Link State Routing protocols provide greater flexibility and sophistication than the Distance Vector routing protocols. They reduce overall broadcast traffic and make better decisions about routing by taking characteristics such as bandwidth, delay, reliability, and load into consideration, instead of basing their decisions solely on distance or hop count. In this Project only delay characteristic of routing is used to implement. But it is possible to use other features on routing just adding some other characteristics to Link class.

In this project , I have seen that link state routing adapts any change such as failure or recovery in the network and update the routing table quickly. On the other hand, it uses intensive resources(memory and CPU) to distribute packages to all other than itself.

- Advantages of Link State:
  - Small routing tables
  - Quick respond for any changes
  - Scalable
- Disadvantages of Link State:
  - Significant CPU
  - Intensive resource(especially for large networks)
  - Hard to implement(I've spent almost 4-full day to implement)

References

1. Andrew Tanenbaum – Computer Networks 5th Edition
2. http://www.javvin.com/routing-protocols.html
3. http://technet.microsoft.com/en-us/library/cc940461.aspx