Comp 5630: Data Communication I

Programming Assignment 2

Distributed Asynchronous Distance Vector Routing

Jigar Makwana  
01711370

**Distance-Vector-Routing**

PROGRAM DESIGN:

• This is a programming assignment to implement a distributed asynchronous distance vector routing algorithm for the given network topology.

• This implementation is done using JAVA programming language.

• The Bellman Ford Algorithm is used to implement distance vector routing algorithm in this project.

• This program finds the minimum distance to all nodes in a graph for each node.

• Each node holds the minimum distance to every other node.

• The nodes respond to change and will update minimum paths if a new one is found. In case of an update, a node will inform all its direct neighbors about it.

• The distance vector routing algorithm is executed in an emulated environment.

• Main file – Project.java and other class files - Entity.java, Event.java, EventList.java, EventListImpl.java, NetworkSimulator.java, Packet.java and Subclasses Entity0.java, Entity1.java, Entity2.java, Entity3.java are needed for the routing algorithm to be executed.

• NetworkSimulator.java represents an emulated network environment.

• It produces the network layer 2, and the transmission and delivery between connected nodes. At the start of the emulated environment it calls on Entity0.java, Entity1.java, Entity2.java, Entity3.java

• The emulated environment calls the Entity (0,1,2,3) subclasses connected to the 4 nodes Each of the nodes contain the following subclasses:

o Entity0(), Entity1(), Entity2(), Entity3() :

These constructors will initialize the corresponding nodes with link costs and adds neighboring nodes. If links change, new packets are created and sent to all neighboring nodes.

o Update (packet p) : In the update() method, the neighboring nodes will receive the new packets from the source node and these values are updated within each of the destination nodes.

**Updated Code**

**Entity0.java**

import java.util.ArrayList;

public class Entity0 extends Entity

{

// Perform any necessary initialization in the constructor

private static final ArrayList<Integer> neighbors = new ArrayList<>();

private int[] minCostEntity0 = new int[4];

public Entity0()

{

System.out.println("Initializing entity 0 \n");

// Adds all neighbors of node 0

neighbors.add(1);

neighbors.add(2);

neighbors.add(3);

int[] initialCosts = new int[]{0, 1, 3, 7}; //Initial Costs declaration

//Link costs between each node is initialized and sent to the distance table.

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++) {

for (int j = 0; j < 4; j++) {

if (i == j) {

distanceTable[i][j] = initialCosts[i];

}

else {

distanceTable[i][j] = 999;

}

}

}

printDT();

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++){

minCostEntity0[i] = caluclateMinCostRecursive(distanceTable[i]);

}

//If next node changes, a new packet with new minimum cost values is created.

//sent to all neighboring nodes.

int k = 0;

for (int j=0 ; j < neighbors.size(); j++) {

Packet p = new Packet(k, neighbors.get(j), minCostEntity0);

NetworkSimulator.toLayer2(p);

}

}

// Handle updates when a packet is received. Students will need to call

// NetworkSimulator.toLayer2() with new packets based upon what they

// send to update. Be careful to construct the source and destination of

// the packet correctly. Read the warning in NetworkSimulator.java for more

// details.

public void update(Packet p)

{

int entityId = 0; //Name of the node is 0

int source = p.getSource(); //get the source ID

int[] currentMinCost = new int[4];

for (int m = 0; m < NetworkSimulator.NUMENTITIES; m++){

currentMinCost[m] = p.getMincost(m);

}

System.out.println(String.format("Entity %d update is called, dest = %d, source = %d",

entityId, p.getDest(), p.getSource()));

System.out.println("costs: ");

for (int z = 0; z < 4; z++){

System.out.println(p.getMincost(z));

}

System.out.println("\n");

boolean isUpdated = false;

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++) {

int updatableValue = this.distanceTable[source][source] + currentMinCost[i];

if (updatableValue < 999) {

this.distanceTable[i][source] = updatableValue;

} else {

this.distanceTable[i][source] = 999;

}

}

printDT();

//calculate the minimum cost between node 0 and its neighboring nodes

int[] cost;

cost = minCostEntity0;

for (int j = 0; j < NetworkSimulator.NUMENTITIES; j++) {

minCostEntity0[j] = caluclateMinCostRecursive(this.distanceTable[j]);

}

for (int k = 0; k < 4; k++) {

if (cost[k] != minCostEntity0[k]) {

isUpdated = true; //Assert isUpdated to true.

}

}

//After updating the costs, further the

// distance table is printed with the new updated minimum costs.

if (isUpdated) {

for (int j : neighbors) {

Packet updatedPacket = new Packet(entityId, j, minCostEntity0);

NetworkSimulator.toLayer2(updatedPacket);

}

}

else{

System.out.println("no updates");

}

}

//method to calculate the minimum cost

protected static int caluclateMinCostRecursive ( int a[] ) {

int min = a[0];

for (int m: a){

if (m < min){

min = m;

}

}

return min;

}

public void linkCostChangeHandler(int whichLink, int newCost)

{

}

//print the distance table

public void printDT()

{

System.out.println();

System.out.println(" via");

System.out.println(" D0 | 1 2 3");

System.out.println("----+------------");

for (int i = 1; i < NetworkSimulator.NUMENTITIES; i++)

{

System.out.print(" " + i + "|");

for (int j = 1; j < NetworkSimulator.NUMENTITIES; j++)

{

if (this.distanceTable[i][j] < 10)

{

System.out.print(" ");

}

else if (this.distanceTable[i][j] < 100)

{

System.out.print(" ");

}

else

{

System.out.print(" ");

}

System.out.print(this.distanceTable[i][j]);

}

System.out.println();

}

}

//Method to print the minimum costs for node 0

public void printMinCost() {

System.out.print("Minimum costs for Entity 0: ");

for (int l = 0 ; l < 4; l++){

System.out.print(minCostEntity0[l]);

}

System.out.println("\n");

}

}

**Entity1.java**

import java.util.ArrayList;

public class Entity1 extends Entity

{

private static final ArrayList<Integer> neighbors = new ArrayList<>();

private int[] minCostEntity1 = new int[4];

// Perform any necessary initialization in the constructor

public Entity1()

{

System.out.println("\nInitializing entity 1 ");

int[] initialCosts = new int[]{1, 0, 1, 999}; //Initial Costs declaration

// Adds all neighbors of node 0

neighbors.add(0);

neighbors.add(2);

//Link costs between each node is initialized and sent to the distance table.

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++) {

for (int j = 0; j < 4; j++) {

if (i == j) {

this.distanceTable[i][j] = initialCosts[i];

}

else {

this.distanceTable[i][j] = 999;

}

}

}

printDT();

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++){

minCostEntity1[i] = caluclateMinCostRecursive(this.distanceTable[i]);

}

//If next node changes, a new packet with new minimum cost values is created.

//sent to all neighboring nodes.

int k = 1;

for (int j : neighbors){

Packet p = new Packet(k, j, minCostEntity1);

NetworkSimulator.toLayer2(p);

}

}

// Handle updates when a packet is received. Students will need to call

// NetworkSimulator.toLayer2() with new packets based upon what they

// send to update. Be careful to construct the source and destination of

// the packet correctly. Read the warning in NetworkSimulator.java for more

// details.

public void update(Packet p)

{

int entityId = 1; //Name of the node is 1

int source = p.getSource(); //get the source ID

int[] currentMinCost = new int[4];

for (int m = 0; m < NetworkSimulator.NUMENTITIES; m++){

currentMinCost[m] = p.getMincost(m);

}

System.out.println(String.format("Entity %d update is called, dest = %d, source = %d",

entityId, p.getDest(), p.getSource()));

System.out.println("costs: ");

for (int z =0; z < currentMinCost.length; z++){

System.out.println(currentMinCost[z]);

}

System.out.println("\n");

boolean isUpdated = false;

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++) {

int updatableValue = this.distanceTable[source][source] + currentMinCost[i];

if (updatableValue < 999) {

this.distanceTable[i][source] = updatableValue;

} else {

this.distanceTable[i][source] = 999;

}

}

this.printDT();

//calculate the minimum cost between node 1 and its neighboring nodes

int[] cost;

cost = minCostEntity1;

for (int j = 0; j < NetworkSimulator.NUMENTITIES; j++) {

minCostEntity1[j] = caluclateMinCostRecursive(this.distanceTable[j]);

}

for (int k = 0; k < 4; k++) {

if (cost[k] != minCostEntity1[k]) {

isUpdated = true; //Assert isUpdated to true.

}

}

//After updating the costs, further the

// distance table is printed with the new updated minimum costs.

if (isUpdated) {

for (int j : neighbors) {

Packet updatedPacket = new Packet(entityId, j, minCostEntity1);

NetworkSimulator.toLayer2(updatedPacket);

}

}

else{

System.out.println("no updates");

}

}

public void linkCostChangeHandler(int whichLink, int newCost)

{

}

public void printDT()

{

System.out.println();

System.out.println(" via");

System.out.println(" D1 | 0 2");

System.out.println("----+--------");

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++)

{

if (i == 1)

{

continue;

}

System.out.print(" " + i + "|");

for (int j = 0; j < NetworkSimulator.NUMENTITIES; j += 2)

{

if (this.distanceTable[i][j] < 10)

{

System.out.print(" ");

}

else if (this.distanceTable[i][j] < 100)

{

System.out.print(" ");

}

else

{

System.out.print(" ");

}

System.out.print(this.distanceTable[i][j]);

}

System.out.println();

}

}

//method to calculate the minimum cost

protected static int caluclateMinCostRecursive ( int a[] ) {

int min = a[0];

for (int m: a){

if (m < min){

min = m;

}

}

return min;

}

//Method to print the minimum costs for node 1

public void printMinCost() {

System.out.print("Minimum costs for Entity 1: ");

for (int l = 0; l < 4; l++) {

System.out.print(minCostEntity1[l]);

}

System.out.println("\n");

}

}

**Entity2.java**

import java.util.ArrayList;

public class Entity2 extends Entity

{

private static final ArrayList<Integer> neighbors = new ArrayList<>();

private int[] minCostEntity2 = new int[4];

// Perform any necessary initialization in the constructor

public Entity2()

{

System.out.println("\nInitializing entity 2 ");

int[] initialCosts = new int[]{3, 1, 0, 2}; //Initial Costs declaration

// Adds all neighbors of node 2

neighbors.add(0);

neighbors.add(1);

neighbors.add(3);

//Link costs between each node is initialized and sent to the distance table.

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++) {

for (int j = 0; j < 4; j++) {

if (i == j) {

this.distanceTable[i][j] = initialCosts[i];

}

else {

this.distanceTable[i][j] = 999;

}

}

}

printDT();

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++){

minCostEntity2[i] = caluclateMinCostRecursive(this.distanceTable[i]);

}

//If next node changes, a new packet with new minimum cost values is created.

//sent to all neighboring nodes.

int k = 2;

for (int j=0 ; j < neighbors.size(); j++){

Packet p = new Packet(k, neighbors.get(j), minCostEntity2);

NetworkSimulator.toLayer2(p);

}

}

// Handle updates when a packet is received. Students will need to call

// NetworkSimulator.toLayer2() with new packets based upon what they

// send to update. Be careful to construct the source and destination of

// the packet correctly. Read the warning in NetworkSimulator.java for more

// details.

public void update(Packet p)

{

int entityId = 2; //Name of the node is 2

int source = p.getSource(); //get the source ID

int[] currentMinCost = new int[4];

for (int m = 0; m < NetworkSimulator.NUMENTITIES; m++){

currentMinCost[m] = p.getMincost(m);

}

System.out.println(String.format("Entity %d update is called, dest = %d, source = %d",

entityId, p.getDest(), p.getSource()));

boolean isUpdated = false;

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++) {

int updatableValue = this.distanceTable[source][source] + currentMinCost[i];

if (updatableValue < 999) {

this.distanceTable[i][source] = updatableValue;

} else {

this.distanceTable[i][source] = 999;

}

}

printDT();

//calculate the minimum cost between node 2 and its neighboring nodes

int[] cost;

cost = minCostEntity2;

for (int j = 0; j < NetworkSimulator.NUMENTITIES; j++) {

minCostEntity2[j] = caluclateMinCostRecursive(this.distanceTable[j]);

}

for (int k = 0; k < 4; k++) {

if (cost[k] != minCostEntity2[k]) {

isUpdated = true; //Assert isUpdated to true.

}

}

//After updating the costs, further the

// distance table is printed with the new updated minimum costs.

if (isUpdated) {

for (int j : neighbors) {

Packet updatedPacket = new Packet(entityId, j, minCostEntity2);

NetworkSimulator.toLayer2(updatedPacket);

}

}

else{

System.out.println("no updates");

}

}

public void linkCostChangeHandler(int whichLink, int newCost)

{

}

public void printDT()

{

System.out.println();

System.out.println(" via");

System.out.println(" D2 | 0 1 3");

System.out.println("----+------------");

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++)

{

if (i == 2)

{

continue;

}

System.out.print(" " + i + "|");

for (int j = 0; j < NetworkSimulator.NUMENTITIES; j++)

{

if (j == 2)

{

continue;

}

if (this.distanceTable[i][j] < 10)

{

System.out.print(" ");

}

else if (this.distanceTable[i][j] < 100)

{

System.out.print(" ");

}

else

{

System.out.print(" ");

}

System.out.print(this.distanceTable[i][j]);

}

System.out.println();

}

}

//method to calculate the minimum cost

protected static int caluclateMinCostRecursive ( int a[] ) {

int min = a[0];

for (int m: a){

if (m < min){

min = m;

}

}

return min;

}

//Method to print the minimum costs for node 2

public void printMinCost() {

System.out.print("Minimum costs for Entity 2: ");

for (int l = 0 ; l < 4; l++){

System.out.print(minCostEntity2[l]);

}

System.out.println("\n");

}

}

**Entity3.java**

import java.util.ArrayList;

public class Entity3 extends Entity

{

private static final ArrayList<Integer> neighbors = new ArrayList<>();

private int[] minCostEntity3 = new int[4];

// Perform any necessary initialization in the constructor

public Entity3()

{

System.out.println("\nInitializing entity 3 ");

int[] initialCosts = new int[]{7, 999, 2, 0}; //Initial Costs declaration

// Adds all neighbors of node 3

neighbors.add(2);

neighbors.add(0);

//Link costs between each node is initialized and sent to the distance table.

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++) {

for (int j = 0; j < 4; j++) {

if (i == j) {

this.distanceTable[i][j] = initialCosts[i];

}

else {

this.distanceTable[i][j] = 999;

}

}

}

printDT();

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++){

minCostEntity3[i] = caluclateMinCostRecursive(this.distanceTable[i]);

}

//If next node changes, a new packet with new minimum cost values is created.

//sent to all neighboring nodes.

int k = 3;

for (int j : neighbors){

Packet p = new Packet(k, j, minCostEntity3);

NetworkSimulator.toLayer2(p);

}

}

// Handle updates when a packet is received. Students will need to call

// NetworkSimulator.toLayer2() with new packets based upon what they

// send to update. Be careful to construct the source and destination of

// the packet correctly. Read the warning in NetworkSimulator.java for more

// details.

public void update(Packet p)

{

int entityId = 3; //Name of the node is 3

int source = p.getSource(); //get the source ID

int[] currentMinCost = new int[4];

for (int m = 0; m < NetworkSimulator.NUMENTITIES; m++){

currentMinCost[m] = p.getMincost(m);

}

System.out.println(String.format("Entity %d update is called, dest = %d, source = %d",

entityId, p.getDest(), p.getSource()));

System.out.println("costs: ");

for (int z = 0; z < currentMinCost.length; z++){

System.out.println(currentMinCost[z]);

}

System.out.println("\n");

boolean isUpdated = false;

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++) {

int updatableValue = this.distanceTable[source][source] + currentMinCost[i];

if (updatableValue < 999) {

distanceTable[i][source] = updatableValue;

} else {

distanceTable[i][source] = 999;

}

}

printDT();

//calculate the minimum cost between node 3 and its neighboring nodes

int[] cost;

cost = minCostEntity3;

for (int j = 0; j < NetworkSimulator.NUMENTITIES; j++) {

minCostEntity3[j] = caluclateMinCostRecursive(this.distanceTable[j]);

}

for (int k = 0; k < 4; k++) {

if (cost[k] != minCostEntity3[k]) {

isUpdated = true; //Assert isUpdated to true.

}

}

//After updating the costs, further the

// distance table is printed with the new updated minimum costs.

if (isUpdated) {

for (int j : neighbors) {

Packet updatedPacket = new Packet(entityId, j, minCostEntity3);

NetworkSimulator.toLayer2(updatedPacket);

}

}

else{

System.out.println("no updates");

}

}

public void linkCostChangeHandler(int whichLink, int newCost)

{

}

public void printDT()

{

System.out.println(" via");

System.out.println(" D3 | 0 2");

System.out.println("----+--------");

for (int i = 0; i < NetworkSimulator.NUMENTITIES; i++)

{

if (i == 3)

{

continue;

}

System.out.print(" " + i + "|");

for (int j = 0; j < NetworkSimulator.NUMENTITIES; j += 2)

{

if (this.distanceTable[i][j] < 10)

{

System.out.print(" ");

}

else if (this.distanceTable[i][j] < 100)

{

System.out.print(" ");

}

else

{

System.out.print(" ");

}

System.out.print(this.distanceTable[i][j]);

}

System.out.println();

}

}

//method to calculate the minimum cost

protected static int caluclateMinCostRecursive ( int a[] ) {

int min = a[0];

for (int m: a){

if (m <= min){

min = m;

}

}

return min;

}

//Method to print the minimum costs for node 3

public void printMinCost() {

System.out.print("Minimum costs for Entity 3: ");

for (int l = 0 ; l < 4; l++){

System.out.print(minCostEntity3[l]);

}

System.out.println("\n");

}

}

**Result:**

Machine generated alternative text:
Assignments\programAssn#2\Assn 2\out\production\Assn 2 
Project 
Network Simulator VI. O 
Enter trace level (>= 0) : 
Will the link change (1 — 
Yes, 0 — 
Enter random seed: [random] 2 
Initializing entity 0 
via 
No) : 
DO I 
Il 
1 
1 
2 
999 
3 
999 
3 
999 
999 
21 999 
31 999 
Initializing entity 1 
via 
DI I 
01 
O 
1 
21 999 
31 999 
2 
999 
1 
999 
Initializing entity 2 
01 
0 
3 
Il 999 
31 999 
via 
1 
999 
1 
999 
3 
999 
999 
2 
Initializing entity 3 
via 
01 
O 
7 
Il 999 
21 999 
2 
999 
999 
2 
main ( ) : 
src=2 , 
event received. t=l. 
dest=3, contents= C 3, 
6731444532100161, 
node=3 

Machine generated alternative text:
o 
7 
Il 999 
21 999 
no updates 
Entity 
3, 
costs: 
3 
1 
2 
D3 1 
01 
3 
update 
via 
is 
called, 
dest = 
source = 
2 
2 
5 
3 
2 
main(): event received. t=3.0534213733607594, 
src=2, dest=l, contents=[3, 1, O, 2] 
Entity I update is called, dest = 
I, source = 2 
node—I 
costs: 
3 
1 
2 
DI I 
01 
via 
1 
21 999 
31 999 
no updates 
2 
4 
1 
3 
main(): event received. t=5.471403340878017, 
src=O, dest=3, 1, 3, 7] 
Entity 3 update is called, dest — 
3, source = 
costs: 
node=3 
1 
3 
7 
D3 
via 
o 
2 

Machine generated alternative text:
01 
Il 
21 
7 
8 
10 
5 
3 
2 
no updates 
main(): event received. .580322424179152, 
src=0, dest=l, contents=C0, 1, 3, 7] 
Entity 1 update is called, dest = 
1, source 
costs: 
1 
3 
7 
DI I 
01 
21 
31 
via 
o 
1 
4 
8 
2 
4 
1 
3 
no updates 
main(): event received. . 688419464619472, 
src=3, dest=2, contents=[7, 999, 2, 0 
Entity 2 update is called, dest — 
2 , source — 
01 
o 
3 
Il 999 
31 999 
no updates 
via 
1 
999 
1 
999 
3 
9 
999 
2 
src=l, dest=2, 0, 1, 999] 
Entity 2 update is called, dest 
2 , source — 
node=l 
node=2 
3 
node—2 
1 
01 
Il 
31 
3 
999 
999 
via 
1 
2 
1 
999 
3 
9 
999 
2 

Machine generated alternative text:
no updates 
main(): event received. t=9.11302861627049, 
src=O, dest=2, contents=[O, I, 3, 7] 
Entity 2 update is called, dest — 
2 , source = 
01 
Il 
31 
3 
4 
via 
1 
2 
1 
10 999 
3 
9 
999 
2 
no updates 
node—2 
node=O 
main(): event received. t=9.546249434110536, 
src=3, dest=O, contents=[7, 999, 2, O] 
Entity O update is called, dest 
O, source = 3 
costs: 
7 
999 
2 
DO I 
Il 
via 
1 
1 
21 999 
31 999 
2 
999 
3 
999 
3 
999 
9 
7 
no updates 
main ( ) : event 
received. t-9.872892399125815, 
src=l, dest=O, O, I, 9991 
node=O 
Entity O update is called, dest = 
O, source = 1 
costs: 
1 
1 
999 
DO 
1 
via 
2 
3 

Machine generated alternative text:
Il 
21 
1 
2 
999 
3 
999 
999 
9 
31 999 
no updates 
main(): event 
received. t-9.88678750508584, 
src=2, dest=O, contents=[3, 1, O, 2] 
node=O 
Entity O update is called, dest = 
O, source = 2 
costs: 
3 
1 
2 
Il 
21 
via 
1 
1 
2 
2 
4 
3 
5 
3 
999 
9 
7 
31 999 
no updates 
Simulator terminated at t=9. 88678750508584, 
no packets in medium. 
Minimum 
Minimum 
Minimum 
Minimum 
Process 
costs 
costs 
costs 
costs 
for Entity O: 
for Entity 1: 
for Entity 2 : 
for Entity 3: 
finished with exit 
0125 
1013 
2102 
5320 
code 0 