# ANEXO I

**class cBackgroundPanel**

```
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Line2D;//Dibujando linea en Java

import javax.swing.JPanel;

public class cBackgroundPanel extends JPanel {

	cNetwork network;
	String type;

  public cBackgroundPanel(cNetwork net, String type) {
	network = net;
	this.type = type;

  }

	protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    Graphics2D g2 = (Graphics2D) g;
    Graphics2D g3 = (Graphics2D) g;
    g2.setColor(Color.BLUE);
    g2.setStroke(new BasicStroke(1));
    int x, y, r;
    int x1,y1,x2,y2;

    if (type == "regular") {
	for (int i=0; i<network.numberOfWSNNodes; i++) {
		r = network.nodes.get(i).radio;
		x = network.nodes.get(i).posx;
		y = network.nodes.get(i).posy;
		g2.drawOval(x-r, y-r,2*r ,2*r );
		//g3.drawLine(x, y,2*r ,2*r);
		//g3.drawLine(network.nodes.get(i).posx, network.nodes.get(i).posy, network.nodes.get(i).posx,
network.nodes.get(i).posy);
	}
    }

    else if(type == ""){
	int X[] = network.dijkstra(network.nodes.get(1), "distance");
	for (int i=0; i<network.numberOfWSNNodes; i++) {
		x1= network.links.get(i).nodeSource.posx;
	y1= network.links.get(i).nodeSource.posy ;
	x2= network.links.get(i).nodeSource.posx ;
	y2= network.links.get(i).nodeSource.posy ;

				if (X[i] > 1){
					//System.out.println("Nodo raíz " + i);
				}
				else{
					System.out.println("Predecesor de " + i + " es " + X[i]);
				}
				g3.drawLine(network.nodes.get(i).posx, network.nodes.get(i).posy,
network.nodes.get(i).posx, network.nodes.get(i).posy);
			}

    }
```

```
    /*
    else {
      for (int i=0; i<network.numberOfWSNNodes; i++) {
          /* x1= network.links.get(i).nodeSource.posx;
          y1= network.links.get(i).nodeSource.posy ;
          x2= network.links.get(i).nodeSource.posx ;
          y2= network.links.get(i).nodeSource.posy ;*/
          //g2.drawOval(x-r, y-r,2*r ,2*r );
          //g3.drawLine(x, y,2*r ,2*r);
                    //cNetworkFrame networkDrawDijkstra = new cNetworkFrame(network);
                    //networkDrawDijkstra.setTitle("Network: " + network.name);

          }

}
```

**class cLink**

```
import java.util.ArrayList;
import java.util.Random;

public class cLink {
        // Identification
        public int linkID;                      // identificador
        public String type;                             // tipo de enlace (core, core-PA, PA-PE)

        // Attributes
        public int level;                       // 1=core-core, 2=core-PA, 3=PA-PE
        public int distance;                    // distancia
        public cNode nodeSource;  // nodo origen
        public cNode nodeDest;                  // nodo destino
        public int nodeSourcePort;// numero de puerto del nodo origen
        public int nodeDestPort;    // numero de puerto del nodo destino

        public short mpssLinkID[];              // ID de los enlaces

        public cLink(cNode source, cNode dest, String type, int level, int distance) {
                this.type = type;
                this.level = level;
                this.distance = distance;
                nodeSource = source;
                nodeDest = dest;
                nodeSource.connect(this, "out");
                nodeDest.connect(this, "in");
        }

}
```

**class cMulticastGroup**

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class cMulticastGroup {

        int groupID;                                    // ID del grupo
        ArrayList<cNode> members;               // lista de miembros
        //ArrayList<cLink> linksGroupTree; // lista de enlaces árbol de grupos multicast
        cNetwork network;

        public cMulticastGroup(int id, cNetwork network) {
```

```
		this.groupID = id;
		this.network = network;
		members = new ArrayList<cNode>(0);

	}

	public cMulticastGroup(int id,  int membersArray[], cNetwork network) {
		this.groupID = id;
		this.network = network;
		members = new ArrayList<cNode>(0);
		for (int i=0; i<membersArray.length; i++) {
			if (membersArray[i] == 1)
				members.add(network.nodes.get(i));
		}

	}

	public short[] buildChildBF(cNode root, ArrayList<cLink> linksGroupTree, int m) {
		//metodo postorden, comienza a leer desde los nodos hijos hasta llegar al nodo padre

		cNode pivot = root;//pivot es igual que el nodo padre
		short[] summary = new short[m];
		if (network.numChildren(pivot) > 0) {		//si pivot tiene hijos
			for(int i=0; i<pivot.outLinks.size(); i++) {	//comienza desde el nodo 0 hasta el total
de nodos hijos del padre
				if (linksGroupTree.contains(pivot.outLinks.get(i)))//lo define como nodo
padre porque tiene nodos hijos
					summary = MyUtils.OR(summary,
buildChildBF(pivot.outLinks.get(i).nodeDest, linksGroupTree, m));
			}
			//System.out.println(pivot.name);
			pivot.childrenGroupBF = summary;
			return (MyUtils.OR(summary, pivot.mpssNodeID));
		}
		else {	// cuando se llega a un nodo hoja
			//System.out.println(pivot.name);
			return (pivot.mpssNodeID);
		}
	}

	public short[] calculateLabelBF(ArrayList <cLink> linksTree) {
		/* Procedimiento de la creación de la etiqueta de filtro de Bloom*/

		int m = linksTree.get(0).nodeDest.mpssNodeID.length;
		short labelBF[] = new short[m];	// Esta es la etiqueta del filtro Bloom que va a ser
encontrado
		for (int i=0; i<linksTree.size(); i++) {// para cada enlace del linksTree de los grupos multicast
			labelBF = MyUtils.OR(labelBF, linksTree.get(i).nodeDest.mpssNodeID);	// debido a la
propiedad asociativa por la operación OR,labelBF recoge el resultado en cada iteración

		}
		return labelBF;

	}

	public int[] launchPacketMPSS(cNode sourceNode, short labelBF[], ArrayList<cLink> linksGroupTree, int
ttlValue) {
		/* Ahora el paquete con labelBF se inicia en el nodo origen a nodos destinos PE
		 * Calculo de ancho de banda desperdiciado que deberia realizarse
		 */

		ArrayList<cNode> cola = new ArrayList<cNode>(0);	//	creación de la cola
		ArrayList<Boolean> colaPath = new ArrayList<Boolean>(0);
		ArrayList<Integer> colaTTL = new ArrayList<Integer>(0);
```

```
ArrayList<Integer> colaPredIndex = new ArrayList<Integer>(0);
cNode pivot, previousNode;
boolean rightPath, loop;
int ttl, ttlPath, iNextPivot = 0, nColas = 1, auxColas, temp, endPackets = 0;
int m = labelBF.length;

cola.add(sourceNode);
colaPredIndex.add(0);
colaPath.add(true);
colaTTL.add(ttlValue);
pivot = new cNode();
int bwUsed = 0;
int falsePositives = 0;
int bwWasted = 0;
int loopsDetected = 0;

while (nColas > iNextPivot) {

	pivot = cola.get(iNextPivot);
	previousNode = cola.get(colaPredIndex.get(iNextPivot));
	rightPath = colaPath.get(iNextPivot);
	ttlPath = colaTTL.get(iNextPivot);
	auxColas = iNextPivot;
	//		System.out.print("\n" + previousNode.name + " --> " + pivot.name + " --> " );

	if (MyUtils.BloomFilter(labelBF, pivot.childrenGroupBF) &&
MyUtils.fillFactor(pivot.childrenGroupBF)>0) { //Si la comparación es correcta

		for (int i=0; i<pivot.outLinks.size(); i++) {
			cLink auxLink = pivot.outLinks.get(i);  // auxLink es el enlace para ser
testeada ahora
			cNode auxNode = auxLink.nodeDest;
			ttl = ttlPath;
			//			System.out.print("\n		" +
auxLink.nodeDest.name);
			if (previousNode != auxNode) { // Esto es para evitar volver al nodo
predecesor
				if (ttlPath > 0)   { // Esto es para evitar volver al nodo
anterior o al bucle infinito(A pesar que no pasa por el BF)
					if (linksGroupTree.contains(auxLink)) {     // El
enlace es parte del árbol
						if (rightPath == true) {   // Si el camino
estaba siendo correcto
							bwUsed++;
							cola.add(auxNode);
							colaTTL.add(ttlPath-1);
//modificado
							colaPath.add(true);
							colaPredIndex.add(auxColas);
							nColas++;
							//Aki System.out.println("Envío
OK: " + pivot.name + " --> " + auxNode.name);
						}
						else { // Si el camino esta siendo el
incorrecto
							bwWasted++;
							ttl--;

	cola.add(auxNode);//auxLink.nodeDest);
							colaTTL.add(ttlPath-1);
//modificado
							colaPath.add(false);
							colaPredIndex.add(auxColas);
							nColas++;
```

```
                                                                        }
                                                                }
                                                                else  {// Este enlace no es parte del árbol (Falso
positivo)
                                                                        if (rightPath == true) {  // Si el camino
esta siendo correcto hasta ahora
                                                                                falsePositives++;
                                                                                //AKI
System.out.println("Paquetes perdidos por radiación: " + pivot.name + "-->" + auxNode.name);
                                                                                //int g=falsePositives;
                                                                                //System.out.println("Numero
de Falsos Positivos :" + g);
                                                                        }
                                                                        else { // Si el camino estaba siendo
incorrecto
                                                                                bwWasted++;
                                                                                //AKI System.out.println("
Paquete desperdiciado: " + pivot.name + "-->" + auxNode.name);
                                                                        }
                                                                        ttl--;
                                                                        temp = colaPredIndex.get(iNextPivot);
                                                                        loop = false;
                                                                        while (cola.get(temp) != sourceNode) {
                                                                                if (cola.get(temp) ==
auxNode){//auxLink.nodeDest) {      // Esto va a generar un bucle
                                                                                        loopsDetected++;   //
El numero de bucles se incrementa y no hace nada
                                                                                        // System.out.print("
Loop detectado");
                                                                                        loop = true;
                                                                                        break;
                                                                                }
                                                                                else
                                                                                        temp =
colaPredIndex.get(temp);
                                                                        }
                                                                        if (loop != true) { // Este no va a generar
algun loop y no se ha logrado un nodo hoja(Por lo tanto cortamos el bucle)
                                                                                //}
                                                                                //else {
                                                                                cola.add(auxNode);
                                                                                colaTTL.add(ttlPath-
1);//modificado
                                                                                colaPath.add(false);
                                                                                colaPredIndex.add(auxColas);
                                                                                nColas++;
                                                                        }
                                                                }
                                                                //}
                                                        }
                                                        else { // Si ttlValue < 0 EL recorrido del paquete se suspende
(no deberia suceder,pero es aqui por si a caso)
                                                                //System.out.println(" TTL out");
                                                        }
                                                }
                                        }
                                iNextPivot++;
                        }
                        else
                                iNextPivot++;

                }

                int results[] = new int[7];
```

```
                results[0] = falsePositives;
                results[1] = loopsDetected;
                results[2] = bwWasted;
                results[3] = bwUsed;      // bwUsed = bw useful
                //  results[4] = ( (this.destPEs.size() - 1) - endPackets) + (linksTree.size() - bwUsed ); //
Comprueba errores, 1: paquetes que no llegaron a destPEs, 2: tree == bwUsed
                results[6] = m * (bwUsed + falsePositives + bwWasted) / 8;  // Calculo de la cabecera de
sobrecarga (en numero de bytes): suma de enlances donde el paquete a atravesado

                return results;
        }
}
```

**class cNetwork**

```
import java.util.ArrayList;

public class cNetwork
{

        // identificación
        String name;                    // Nombre de la red
        int m;

        // matrices
        int[][] pos;                    // matriz de x,y de cada nodo en la interfaz gráfica
        int[] PEs;
        int[][] neighbors;          // matriz de nodos por nodos (Para SPF algoritmos, etc.)

        // componentes
        ArrayList<cNode> nodes;   // nodo central
        ArrayList<cLink> links;       // enlace central
        ArrayList<cLink> linksTree; // enlaces del árbol Dijkstra

        int radio;
        int numberOfWSNNodes;
        int numberOfCoreLinks;
        int totalNumberOfLinks;

        // interfaz gráfica;
        cNetworkFrame networkDraw;

        public cNetwork (String name, int m, int radio)
        {
                this.name = name;
                this.m = m;
                this.radio = radio;
                switch (name)
                {
                case "WArequipa":
                {
                        // nodo central y enlace central
                        pos = new int[8][2];
                        pos[0][0] = 100; pos[0][1] = 100;//R1
                        pos[1][0] = 90; pos[1][1] = 180;//R2
                        pos[2][0] = 180; pos[2][1] = 60;//R3
                        pos[3][0] = 160; pos[3][1] = 140;//R4
                        pos[4][0] = 260; pos[4][1] = 75;//R5
                        pos[5][0] = 220; pos[5][1] = 160;//R6
                        pos[6][0] = 150; pos[6][1] = 220;//R7
                        pos[7][0] = 340; pos[7][1] = 105;//R8
                        String cities[] = {"R1","R2","R3","R4","R5","R6","R7","R8"};
                        build(pos, cities);
```

```
                    break;
                }
                case "WPeru":
                {
                    // nodo central y enlace central
                    pos = new int[14][2];
                    pos[0][0] = 100; pos[0][1] = 100;//R1
                    pos[1][0] = 90; pos[1][1] = 180;//R2
                    pos[2][0] = 180; pos[2][1] = 60;//R3
                    pos[3][0] = 160; pos[3][1] = 140;//R4
                    pos[4][0] = 260; pos[4][1] = 75;//R5
                    pos[5][0] = 220; pos[5][1] = 160;//R6
                    pos[6][0] = 150; pos[6][1] = 220;//R7
                    pos[7][0] = 340; pos[7][1] = 105;//R8
                    pos[8][0] = 305; pos[8][1] = 180;//R9
                    pos[9][0] = 230; pos[9][1] = 250;//R10
                    pos[10][0] = 390; pos[10][1] = 170;//R11
                    pos[11][0] = 340; pos[11][1] = 240;//R12
                    pos[12][0] = 305; pos[12][1] = 310;//R13
                    pos[13][0] = 410; pos[13][1] = 285;//R14
                    String cities[] =
{"R1","R2","R3","R4","R5","R6","R7","R8","R9","R10","R11","R12","R13","R14"};
                    build(pos, cities);
                    break;
                }
                case "Test":
                {
                    // nodo núcleo y enlace núcleo
                    pos = new int[5][2];
                    pos[0][0] = 280; pos[0][1] = 50;
                    pos[1][0] = 350; pos[1][1] = 70;
                    pos[2][0] = 240; pos[2][1] = 100;
                    pos[3][0] = 300; pos[3][1] = 150;
                    pos[4][0] = 190; pos[4][1] = 180;
                    String cities[] = {"R1","R2","R3","R4","R5"};
                    build(pos, cities);
                    break;
                }
                case "Prueba":
                {
                    // nodo núcleo y enlace núcleo
                    pos = new int[5][2];
                    pos[0][0] = 200; pos[0][1] = 60;
                    pos[1][0] = 280; pos[1][1] = 40;
                    pos[2][0] = 450; pos[2][1] = 60;
                    pos[3][0] = 150; pos[3][1] = 70;
                    pos[4][0] = 370; pos[4][1] = 40;
                    String cities[] = {"R1","R2","R3","R4","R5"};
                    build(pos, cities);
                    break;
                }
                case "Abilene":
                {
                    // nodo núcleo y enlace núcleo
                    pos = new int[10][2];
                    pos[0][0] = 145; pos[0][1] = 700-454;
                    pos[1][0] = 190; pos[1][1] = 700-470;
                    pos[2][0] = 390; pos[2][1] = 700-390;
                    pos[3][0] = 475; pos[3][1] = 700-415;
                    pos[4][0] = 224; pos[4][1] = 700-460;
                    pos[5][0] = 260; pos[5][1] = 700-474;
                    pos[6][0] = 340; pos[6][1] = 700-480;
                    pos[7][0] = 440; pos[7][1] = 700-500;
                    pos[8][0] = 320; pos[8][1] = 700-400;
```

```
                    pos[9][0] = 270; pos[9][1] = 700-540;
                    String cities[] = {"R1","R2","R3","R4","R5","R6","R7","R8","R9","R10"};
                    build(pos, cities);
                    break;
                }
                case "NSFNET":
                {
                    // nodo núcleo y enlace núcleo
                    pos = new int[14][2];
                    pos[0][0] = 234; pos[0][1] = 650-614;
                    pos[1][0] = 360; pos[1][1] = 650-415;
                    pos[2][0] = 478; pos[2][1] = 650-372;
                    pos[3][0] = 617; pos[3][1] = 650-391;
                    pos[4][0] = 780; pos[4][1] = 650-378;
                    pos[5][0] = 853; pos[5][1] = 650-437;
                    pos[6][0] = 992; pos[6][1] = 650-475;
                    pos[7][0] = 1033; pos[7][1] = 650-427;
                    pos[8][0] = 940; pos[8][1] = 650-408;
                    pos[9][0] = 1000; pos[9][1] = 650-381;
                    pos[10][0] = 894; pos[10][1] = 650-219;
                    pos[11][0] = 657; pos[11][1] = 650-132;
                    pos[12][0] = 232; pos[12][1] = 650-252;
                    pos[13][0] = 163; pos[13][1] = 650-377;
                    String cities[] = {"Seattle,WA","Salt Lake
City,UT","Boulder,CO","Lincoln,NE","Champaign,IL","Ann
Arbor,MI","Ithaca,NY","Princeton,NJ","Pittsburgh,PE","College Park,MA","Atlanta, GE","Houston,TX","San
Diego,CA","Palo Alto,CA"};
                    build(pos, cities);
                    break;
                }
                case "COST-266":
                {
                    // Red Cost 266
                    pos = new int[28][2];
                    pos[0][0] = 610; pos[0][1] = 650-452;
                    pos[1][0] = 948; pos[1][1] = 650-86;
                    pos[2][0] = 539; pos[2][1] = 650-188;
                    pos[3][0] = 875; pos[3][1] = 650-259;
                    pos[4][0] = 733; pos[4][1] = 650-450;
                    pos[5][0] = 496; pos[5][1] = 650-279;
                    pos[6][0] = 600; pos[6][1] = 650-414;
                    pos[7][0] = 840; pos[7][1] = 650-330;
                    pos[8][0] = 730; pos[8][1] = 650-534;
                    pos[9][0] = 433; pos[9][1] = 650-507;
                    pos[10][0] = 676; pos[10][1] = 650-397;
                    pos[11][0] = 513; pos[11][1] = 650-585;
                    pos[12][0] = 671; pos[12][1] = 650-461;
                    pos[13][0] = 531; pos[13][1] = 650-442;
                    pos[14][0] = 606; pos[14][1] = 650-285;
                    pos[15][0] = 405; pos[15][1] = 650-163;
                    pos[16][0] = 676; pos[16][1] = 650-277;
                    pos[17][0] = 714; pos[17][1] = 650-351;
                    pos[18][0] = 714; pos[18][1] = 650-640;
                    pos[19][0] = 559; pos[19][1] = 650-375;
                    pos[20][0] = 790; pos[20][1] = 650-365;
                    pos[21][0] = 743; pos[21][1] = 650-175;
                    pos[22][0] = 790; pos[22][1] = 650-609;
                    pos[23][0] = 647; pos[23][1] = 650-357;
                    pos[24][0] = 786; pos[24][1] = 650-340;
                    pos[25][0] = 871; pos[25][1] = 650-450;
                    pos[26][0] = 792; pos[26][1] = 650-269;
                    pos[27][0] = 647; pos[27][1] = 650-314;
```

```
                        String cities[] =
{"Amsterdam","Athens","Barcelona","Belgrade","Berlin","Bordeaux","Brussels","Budapest","Copenhagen","Dublin","
Frankfurt","Glasgow","Hamburg","London","Lyon","Madrid","Milan","Munich","Oslo","Paris","Prague","Rome","Stock
holm","Strasbourg","Vienna","Warsaw","Zagreb","Zurich"};
                        build(pos, cities);
                        break;
                }
                }
        }

        public short[] buildOutterBF(cNode root ) {
                //árbol postorden, comienza a leer desde los nodos hijos hasta llegar el nodo padre
                cNode pivot = root;//pivot es igual que el nodo padre
                short[] summary = new short[m];
                if (this.numChildren(pivot) > 0) {      //si pivot tiene hijos
                        for(int i=0; i<pivot.outLinks.size(); i++) {        //comienza desde el nodo 0 hasta el total
de nodos hijos del padre
                                if (linksTree.contains(pivot.outLinks.get(i)))//lo define como nodo padre
porque tiene nodos hijos
                                        summary = MyUtils.OR(summary,
buildOutterBF(pivot.outLinks.get(i).nodeDest));
                        }
                        //System.out.println(pivot.name);
                        pivot.childrenBF = summary;
                        return (MyUtils.OR(summary, pivot.mpssNodeID));
                }
                else {      // cuando se llega a un nodo hoja
                        //System.out.println(pivot.name);
                        return (pivot.mpssNodeID);
                }
        }

        public short[] buildInnerBF(cNode root, ArrayList<cLink> linksGroupTree, cMulticastGroup group) {
                //tree postorden,comienza a leer desde los nodos hijos hasta llegar el nodo padre
                cNode pivot = root;//pivot es igual que el nodo padre
                short[] summary = new short[m];
                if (this.numChildren(pivot) > 0) {      //si pivot tiene hijos
                        for(int i=0; i<pivot.outLinks.size(); i++) {        //comienza desde el nodo 0 hasta el total
de nodos hijos del padre
                                if (linksGroupTree.contains(pivot.outLinks.get(i)))//lo define como nodo
padre porque tiene nodos hijos
                                        summary = MyUtils.OR(summary,
buildInnerBF(pivot.outLinks.get(i).nodeDest, linksGroupTree, group));
                        }
                        //System.out.println(pivot.name);
                        pivot.addEntry(group, summary);     // añade la entrada con el número de grupos y el
resumen de fwTable del nodo
                        return (MyUtils.OR(summary, pivot.mpssNodeID));
                }
                else {      // cuando se llega a un nodo hoja
                        //System.out.println(pivot.name);
                        return (pivot.mpssNodeID);
                }
        }

        public int numChildren (cNode node) {//numero de hijos contenidos en el nodo padre
                int x = 0;//nodos hijos,contador en cero
                for (int i=0; i<node.outLinks.size(); i++) {//cantidad de nodos hojas
                        if (this.linksTree.contains(node.outLinks.get(i)));//pila con predecesores que
contengan enlaces con otros nodos
                                x++;//si resulta cierto el contador aumenta
                }

                return (x);
```

```
	}

	private void build(int pos[][], String cities[]) {		// metodo que crea instancia a todos los nodos de la WSN

		nodes = new ArrayList<cNode>(0); //inicialización del arreglo/colección de nodos
		links = new ArrayList<cLink>(0); //inicialización del arreglo de enlaces

		/* calculo del numero de nodos y numero de enlaces */
		this.numberOfWSNNodes = pos.length;		// calculo del numero de nodos núcleos

		// creación de nodos
		for (int i=0; i<this.numberOfWSNNodes; i++) {
			nodes.add(new cNode(i+1, m, "R"+Integer.toString(i+1), this.radio,"WSN", cities[i], 20));
			nodes.get(i).posx = pos[i][0];
			nodes.get(i).posy = pos[i][1];
		}

	}

	public void createTableNeighbors() {
		//creación de tabla de vecinos
		int distance;
		neighbors= new int[this.numberOfWSNNodes][this.numberOfWSNNodes];

		for (int i=0; i<this.numberOfWSNNodes; i++) {
			for (int j=i+1; j<numberOfWSNNodes; j++) {
				if (SimTools.isNeighbor(nodes.get(i).posx, nodes.get(i).posy, nodes.get(j).posx, nodes.get(j).posy, nodes.get(i).radio)) {
					distance = SimTools.distance(nodes.get(i).posx, nodes.get(i).posy, nodes.get(j).posx, nodes.get(j).posy);
					links.add(new cLink(nodes.get(i), nodes.get(j), "core", 1, distance));
	// crea el enlace desde i hasta j
					links.add(new cLink(nodes.get(j), nodes.get(i), "core", 1, distance));
	// crea el enlace desde j hasta i
					//System.out.println(numberOfWSNNodes);
					neighbors[i][j] = distance;
					neighbors[j][i] = distance;
				}
			}
		}

		for (int i=0; i<numberOfWSNNodes; i++) {
			System.out.print("Node " + (i+1) + ": ");
			for (int j=0; j<numberOfWSNNodes; j++) {
				if (neighbors[i][j] > 0)
					System.out.print((j+1) + " ");
			}
			System.out.println();
		}

	}

	/* Este metodo encuentra el árbol con la ruta mas corta desde el nodo origen a todos nodo de la red
	 * El nodo origen puede ser algun nodo(PE, PA or core) */
	public int[] dijkstra(cNode sourceNode, String type) {

		int pred[] = new int[this.numberOfWSNNodes];			// inicialización del arreglo de predecesores
		ArrayList<cNode> nodesToAchieve = new ArrayList<cNode>(nodes);	// al comienzo, nodesToAchieve = nodes
		ArrayList<cNode> nodesToEvaluate = new ArrayList<cNode>(0);	// inicialización de la lista nodesEvaluated
```

```
		pred[sourceNode.nodeID-1] = -1;	// El predecesor del origen es el mismo
		cNode pivot = sourceNode; // El origen se inicia como el pivot

		int distances[] = new int[nodes.size()];		// Las matrices/arreglo de distancia desde cada nodo para el origen
		for (int i=0; i<distances.length; i++)  // Todos los valores de la distancia son inicializados como INF
			distances[i] = (int) Double.POSITIVE_INFINITY;
		distances[sourceNode.nodeID-1] = 0;		// La distancia del nodo origen a si mismo es 0
		nodesToAchieve.remove(sourceNode);		// el nodo fuente se elimina desde la lista de nodesToAchieve
		int iPivot, iNewNode, newDist, iMin, min;
		cNode nextPivot;

		// itera mientras todavia quedan nodos para conseguir
		while (!nodesToAchieve.isEmpty()) {
			iPivot = pivot.nodeID-1;
			for (int i=0; i<pivot.outLinks.size(); i++)
				if (nodesToAchieve.contains(pivot.outLinks.get(i).nodeDest)) {  // si el nodo destino no ha sido evaluado
					iNewNode = pivot.outLinks.get(i).nodeDest.nodeID-1;
					newDist= 0;
					// decisión del tipo de metrica (distancia, saltos, etc.)
					if (type.equals("distance"))
						newDist = pivot.outLinks.get(i).distance + distances[iPivot];
	// nueva distancia desde el NewNode al origen,a través del pivot
					else if (type.equals("hops"))
						newDist = 1 + distances[iPivot];		// nueva distancia desde el NewNode al origen,a través del pivot

					if (newDist < distances[iNewNode]) {		// en caso que el recorrido a través del pivot es menor
						distances[iNewNode] = newDist;		// actualiza la matriz de distancias
						pred[iNewNode] = pivot.nodeID-1;	// pivot es ahora el predecesor de el NewNode
					}
					nodesToEvaluate.add(pivot.outLinks.get(i).nodeDest);  // agrega el newNode a la lista de nodesToEvaluate
				}

			/* encuentra el siguiente pivot que provoca la distancia minima al origen */
			min = (int) Double.POSITIVE_INFINITY;		// el primer valor minimo es un valor INF
			iMin = nodesToEvaluate.get(0).nodeID-1;		// indice del primer nodo con la ruta menor al origen
			for (int i=0; i<nodesToEvaluate.size(); i++) {
				iNewNode = nodesToEvaluate.get(i).nodeID-1;
				if (distances[iNewNode] < min) {
					iMin = i;  // indice del primer nodo con la ruta menor al origen
					min = distances[iNewNode];
				}
			}
			nextPivot = nodesToEvaluate.get(iMin);
			/* fin del proceso */

			nodesToEvaluate.remove(nextPivot);		// una vez que el nextPivot ha sido evaluado se elimina desde la lista nodesToEvaluate
			nodesToAchieve.remove(nextPivot); // nextPivot se elimina de la lista nodesToAchieve, porque ya ha sido tratado con exito
			pivot = nextPivot;  // ahora el pivot es el siguiente pivot
		}

		return pred;
```

```
	}

	public cNode getSourceNode(int pred[]) {
	// procedimiento para hallar el nodo origen (sourceNode)
		cNode sourceNode = null;
		for (int r=0; r<this.numberOfWSNNodes; r++)
			if (pred[r] < 0)
				sourceNode = this.nodes.get(r);
		return (sourceNode);
	}

	public void buildLinksTree(int[] pred) {
		/* Este metodo "construye" el árbol, agregando enlaces */

		// procedimiento para hallar el nodo origen (sourceNode)
		cNode sourceNode = null;
		for (int r=0; r<this.numberOfWSNNodes; r++)
			if (pred[r] < 0)
				sourceNode = this.nodes.get(r);

		// procedimiento para crear la lista de enlaces que son parte del árbol
		this.linksTree = new ArrayList<cLink>(0);
		for (int n=0; n<this.numberOfWSNNodes; n++) {		// para cada nodo de la red
			cNode auxNode = this.nodes.get(n);
			while (auxNode != sourceNode) {	// itera desde el nodo PE hasta que se pone al nodo
origen
				int iAuxNode = auxNode.nodeID-1;	// iAuxNode: posición del nodo en la lista
de nodos
				cNode predNode = this.nodes.get(pred[iAuxNode]);	// dado el
pred[iAuxNode], devuelve el predNode
				cLink predAuxLink = auxNode.getLinkFrom(predNode);		// dado el
nodo predecesor, regresa el enlace
				if (!linksTree.contains(predAuxLink))		// añade el enlace al linksTree
					linksTree.add(predAuxLink);
				auxNode = predNode;		// ahora el nuevo auxNode es el predNode
			}
		}

	}

	public void displayNetwork() {
		for (short i=0; i<numberOfWSNNodes; i++)
			nodes.get(i).displayLinks();
	}

	public void displayPredList(int[] pred) {
		// visualización de la lista pred
		for (int i=0; i<pred.length; i++)
			if (pred[i] != -1)
				System.out.println(nodes.get(pred[i]).name + " --> " + nodes.get(i).name);
			else
				System.out.println("Source: " + nodes.get(i).name);
	}

	public void drawNetwork() {
		networkDraw = new cNetworkFrame(this);
		networkDraw.setTitle("Network: " + this.name);
	}

}
```

**class cNetworkFrame**

```
import java.awt.*;
import java.awt.event.*;
import java.net.URL;
import javax.swing.*;

public class cNetworkFrame extends JFrame implements MouseListener {

	cNetwork network;
	String networkInfo;

	cBackgroundPanel background;

	JMenuBar menuBar;
	JMenu menu, submenu;
	JMenuItem item0, item1;

	/*ImageIcon icon;*/
	ImageIcon zcoordinator;
	ImageIcon zrouter;
	ImageIcon zRFD;
	JLabel router[];
	JLabel info;
	int nbOver = 0, nbClick = 0;

	public cNetworkFrame(cNetwork network) {

		this.network = network;

		networkInfo =	network.name + ": " + network.numberOfWSNNodes +" nodes";

		info = new JLabel(networkInfo);

		// creación de la barra menu
		menuBar = new JMenuBar();

		// construccion del primer menu (columna)
		menu = new JMenu("Simulación");
		menu.setMnemonic(KeyEvent.VK_A);
		menu.getAccessibleContext().setAccessibleDescription("Especificaciones de la simulación");
		menuBar.add(menu);

		// crea el grupo de JMenuItems para el primer menu
		item0 = new JMenuItem("Ver información de red", KeyEvent.VK_I);
		item0.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_1, ActionEvent.ALT_MASK));
		item0.getAccessibleContext().setAccessibleDescription("Ver información de red");
		menu.add(item0);

		// crea el grupo de JMenuItems para el primer menu
		item1 = new JMenuItem("Ejecutar", KeyEvent.VK_R);
		item1.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_2, ActionEvent.ALT_MASK));
		item1.getAccessibleContext().setAccessibleDescription("Ejecutar la simulación");
		menu.add(item1);

		// añade JMenuBar a la estructura
		this.setJMenuBar(menuBar);
		item0.addMouseListener(this);
		item1.addMouseListener(this);

		// inicialización de la matriz de rutas núcleo(core routers)
		router = new JLabel[network.numberOfWSNNodes];

		// crea el background, que pinta los enlaces
		background = new cBackgroundPanel(network, "regular");
```

```
		/*// Carga la imagen icon
		icon = createImageIcon("images/router.png", "router");
		int w = icon.getIconWidth();
		int h = icon.getIconHeight();*/

		// carga la imagen coordinator
		zcoordinator = createImageIcon("images/zcoordinator.png", "coordinator");
		int w = zcoordinator.getIconWidth();
		int h = zcoordinator.getIconHeight();

		// carga la imagen router
		zrouter = createImageIcon("images/zrouter.png", "router");
		int w1 = zrouter.getIconWidth();
		int h1 = zrouter.getIconHeight();

		// carga la imagen RFD
		zRFD = createImageIcon("images/zRFD.png", "RFD");
		int w2 = zRFD.getIconWidth();
		int h2 = zRFD.getIconHeight();
		//System.out.println("altura: "+Integer.toString(w));
		//System.out.println("ancho: "+Integer.toString(h));

		background.setLayout(null);

		// Configuración de la estructura
		setContentPane(background);
		background.setBackground(Color.white);
		background.add(info);
		info.setBounds(0,0,1350,20);

		// dibuja los nodos
		for (int i=0; i<network.numberOfWSNNodes; i++) {
			router[i] = new JLabel(network.nodes.get(i).city, zcoordinator, JLabel.CENTER);
			router[i].setName("R"+Integer.toString(i));
			router[i].setFont(new Font("Verdana", Font.PLAIN, 8));
			router[i].addMouseListener(this);
			background.add(router[i]);
			router[i].setBounds(network.nodes.get(i).posx-30/2, network.nodes.get(i).posy-27/2,
30+19, 27);
		}

		/*for (int i=0; i<network.numberOfWSNNodes; i++) {
			router[i] = new JLabel(network.nodes.get(i).city, zrouter, JLabel.CENTER);
			router[i].setName("R"+Integer.toString(i+1));
			router[i].setFont(new Font("Verdana", Font.PLAIN, 8));
			router[i].addMouseListener(this);
			background.add(router[i]);
			router[i].setBounds(network.nodes.get(i).posx, network.nodes.get(i).posy, 20, 18);
		}

		for (int i=0; i<network.numberOfWSNNodes; i++) {
			router[i] = new JLabel(network.nodes.get(i).city, zRFD, JLabel.CENTER);
			router[i].setName("R"+Integer.toString(i+1));
			router[i].setFont(new Font("Verdana", Font.PLAIN, 8));
			router[i].addMouseListener(this);
			background.add(router[i]);
			router[i].setBounds(network.nodes.get(i).posx, network.nodes.get(i).posy, 15, 15);
		}*/

		setSize(1350, 700);
		setVisible(true);

	}
```

```
	/** Devuelve una ImageIcon, o anula si el camino no era valido. */
	protected ImageIcon createImageIcon(String path, String description) {
		URL imgURL = getClass().getResource(path);
		if (imgURL != null) {
			return new ImageIcon(imgURL, description);
		} else {
			System.err.println("No se puede encontrar el archivo: " + path);
			return null;
		}
	}

	public cNode getCoreNode(String coreNodeName) {
		int i = Integer.parseInt(coreNodeName.substring(1))-1;
		return network.nodes.get(i);
	}


	/** metodo para implementar el MouseListener */
	public void mouseClicked(MouseEvent e) {
		// ask if it was a double-click
		if (e.getClickCount() == 2) {
			String n = e.getComponent().getName();
			info.setText(n);
			new cNodeInfoFrame(getCoreNode(n));
		}
	}

	public void mouseEntered(MouseEvent arg0) {
	}

	public void mouseExited(MouseEvent arg0) {
	}

	public void mousePressed(MouseEvent e) {
		if (e.getComponent() == item0) {
			info.setText(networkInfo);
		}
		else if (e.getComponent() == item1) {
			info.setText("Ejecución de Simulación...");
			JFrame frameSimulation = new cSimFrame(network);
		}
	}

	public void mouseReleased(MouseEvent arg0) {
	}

}
```

**class cNode**

```
import java.util.ArrayList;

import javax.swing.table.DefaultTableModel;

public class cNode {

	// Identificador
	public int nodeID;                                  // identificador de nodo único
	public String name;                                 // nombre del nodo
	public String city;                                 // nombre de la ciudad
	public String type;                                 // tipo de nodo (core, PA, PE)

	public int radio;
```

```
        public ArrayList<cLink> inLinks;     // lista de enlaces entrantes al nodo
        public ArrayList<cLink> outLinks;    // lista de enlaces salientes desde nodo

        // Grupos
        ArrayList<Integer> groups;                  // vpnIDs conectados al nodo (solo para nodos PE)
        int numGroupsAttached;                              // numero de VPNs conectados

        public int m;
        public short mpssNodeID[];                  // ID del nodo

        // Puertos
        public int numPorts;                                // número de puertos (entrada o salida)
        public int inPorts[];                               // matriz de puertos de entrada (-1=free,
0..n=indice del enlace)
        public int numUsedInPorts;                          // número de puertos de entrada usados
        public int outPorts[];                              // matriz de puertos de entrada (-1=free,
0..n=indice del enlace)
        public int numUsedOutPorts;                         // número de puertos de entrada usados

        // reenvío de tabla
        public int maxFTSize = 256;                         // tamaño maximo de la tabla de reenvío
        public int tableNumEntries;                         // numero de entradas de la tabla de reenvío
        public short childrenBF[];                  // resumen de nodos hijos
        public short childrenGroupBF[];             // resumen de nodos hijos (para el grupo de árboles, es
temporal)
        public short fwTable[][];                   // tabla de enrutamiento con resúmenes

        // para graficar
        int posx;                                                   // gráfica de cordenadas en x para pintar
el nodo
        int posy;                                                   // gráfica de cordenadas en y para pintar
el nodo

        public cNode () {

        }

        public cNode (int id, int m, String name, int radio, String type, String city, int numberOfPorts) {

                nodeID = id;
                this.name = name;
                this.radio = radio;
                this.type = type;
                this.city = city;
                this.numPorts = numberOfPorts;
                this.m = m;
                this.mpssNodeID = new short[m];
                this.fwTable = new short[this.maxFTSize][m];
                this.numGroupsAttached = 0;

                // inicialización de puertos entrantes
                inPorts = new int[numberOfPorts];
                for (int i=0; i<inPorts.length; i++)    // cada puerto se inicializa con -1 (free)
                        inPorts[i] = -1;
                numUsedInPorts = 0;
                // inicialización de puertos salientes
                outPorts = new int[numberOfPorts];
                for (int i=0; i<outPorts.length; i++)   // cada puerto se inicializa con -1 (free)
                        outPorts[i] = -1;
                numUsedOutPorts = 0;

                inLinks = new ArrayList<cLink>(0);
                outLinks = new ArrayList<cLink>(0);
```

```
                groups = new ArrayList<Integer>(0);
                numGroupsAttached = 0;

        }

        public boolean connect (cLink link, String direction) {

                if (direction == "in") {
                        if (numUsedInPorts < numPorts) {
                                numUsedInPorts++;
                                for (int i=0; i<numPorts; i++) {
                                        if  (inPorts[i] == -1) {                // buscando algun puerto
disponible
                                                inLinks.add(link);
                                                inPorts[i] = inLinks.size();   // asigna el indice del enlace
reciente añadido a la matriz de puertos
                                                return true;
                                        }
                                }
                                return false;
                        }
                        else
                                return false;
                }
                else if (direction == "out") {
                        if (numUsedOutPorts < numPorts) {
                                numUsedOutPorts++;
                                for (int i=0; i<numPorts; i++) {
                                        if  (outPorts[i] == -1) {               // busca algun puerto disponible
                                                outLinks.add(link);
                                                outPorts[i] = outLinks.size();          // asigna el indice del
enlace reciente añadido a la matriz de puertos
                                                return true;
                                        }
                                }
                                return false;
                        }
                        else
                                return false;
                }
                else {
                        System.out.println("No es posible conectar una/un [" + direction + "] enlace en el nodo "
+ this.city + " (NodoID: " + this.nodeID + ")");
                        return false;
                }

        }

        public void displayLinks ()
        {
                System.out.println("Nodo " + name + ": ");
                System.out.println("Enlaces entrantes");
                for (int i=0; i<numPorts; i++)
                        if (inPorts[i] != -1)
                                System.out.println("Desde/A partir: " +
inLinks.get(inPorts[i]).nodeSource.name + " (" + inLinks.get(inPorts[i]).distance + ")");
                System.out.println("Enlaces salientes");
                for (int i=0; i<numPorts; i++)
                        if (outPorts[i] != -1)
                                System.out.println("a/hasta: " + outLinks.get(outPorts[i]).nodeDest.name + " ("
+ outLinks.get(outPorts[i]).distance + ")");
        }
```

```
	public void addEntry(cMulticastGroup group, short[] groupSummary){

		int numGroup = group.groupID - 1;
		this.fwTable[numGroup] = groupSummary;
		this.groups.add(numGroup);

	}

	public DefaultTableModel getNodeGeneralTable() {
		String data[][] = {{"Nodo ID", nodeID+""},
					{"Nombre", name},
					{"Ciudad", city},
					{"Tipo de jerarquía", type},
					{"Número de puertos", numPorts+""},
					{"Número de puertos de entrada utilizados", numUsedInPorts+""},
					{"Número de puertos de salida utilizados", numUsedOutPorts+""},
					{"Reenvío de tablas num. entradas", tableNumEntries+""},
					{"Filtro de Bloom 1",
Integer.toBinaryString(0b1000000000000000).substring(1, 17)},
					{"Filtro de Bloom 2",
Integer.toBinaryString(0b1000000000000000).substring(1, 17)}};
		String col[] = {"Property","Value"};
		DefaultTableModel model = new DefaultTableModel(data,col);
		return model;
	}

	public DefaultTableModel getNodeRoutingTable() {
		String data[][] = {{"0", "001001", "pop()", "1"},
					{"0", "100110", "pop()", "2"},
					{"0", "010000", "pop()", "1"},
					{"0", "010000", "pop()", "2"}};
		String col[] = {"in port","label", "operation", "out port"};
		DefaultTableModel model = new DefaultTableModel(data,col);
		return model;
	}

	public ArrayList<cNode> getConnectedCoreNodes() {
		ArrayList<cNode> aux = new ArrayList<cNode>(0);
		if (this.type == "Core") {
			for (int i=0; i<outLinks.size(); i++)
				if (outLinks.get(i).type == "core")
					aux.add(outLinks.get(i).nodeDest);
			return aux;
		}
		else {
			System.out.println("No es posible llegar a conectar a un nodo núcleo desde " + this.name
+ " (ID Nodo: " + this.nodeID + "), porque no es un nodo núcleo");
			return null;
		}
	}

	/* Metodo que, dado un nodo, devuelve el enlace desde predNode aeste nodo */
	public cLink getLinkFrom(cNode predNode) {

		for (int i=0; i<inLinks.size(); i++)	// revisa los enlaces entrantes
			if (inLinks.get(i).nodeSource == predNode)	// Si el predNode == nodeSource del
enlace
				return inLinks.get(i);		// devuelve el enlace

		return null;

	}

}
```

**class cNodeInfoFrame**

```
import java.awt.*;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.util.ArrayList;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.JTableHeader;

public class cNodeInfoFrame extends JFrame {

	private static final String Dimension = null;
	JLabel info;
	JPanel background;

	public cNodeInfoFrame(cNode coreNode) {

		setTitle("Nodo núcleo " + coreNode.nodeID + " información ... Ciudad: " + coreNode.city);

		JTabbedPane tabs = new JTabbedPane();
		this.add(tabs, BorderLayout.CENTER);

		// tab "Core Router"
		JPanel panel1 = new JPanel();
		tabs.addTab("Ruta núcleo", panel1);

		JTable table1 = new JTable(coreNode.getNodeGeneralTable());
		table1.getTableHeader().setBackground(Color.yellow);
		table1.setRowHeight(20);
		Dimension dim1 = new Dimension(20,1);
		table1.setIntercellSpacing(new Dimension(dim1));
		table1.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
		table1.getColumnModel().getColumn(0).setPreferredWidth(200);
		table1.getColumnModel().getColumn(1).setPreferredWidth(400);
		JScrollPane pane1 = new JScrollPane(table1);
		panel1.add(table1.getTableHeader());
		panel1.add(table1);
		panel1.add(pane1);

		JTable table2 = new JTable(coreNode.getNodeRoutingTable());
		table2.getTableHeader().setBackground(Color.yellow);
		table2.setRowHeight(20);
		Dimension dim2 = new Dimension(20,1);
		table2.setIntercellSpacing(new Dimension(dim2));
		table2.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
		table2.getColumnModel().getColumn(0).setPreferredWidth(60);
		table2.getColumnModel().getColumn(1).setPreferredWidth(200);
		table2.getColumnModel().getColumn(2).setPreferredWidth(100);
		table2.getColumnModel().getColumn(3).setPreferredWidth(60);
		JScrollPane pane2 = new JScrollPane(table2);
		panel1.add(table2.getTableHeader());
		panel1.add(table2);
		panel1.add(pane2);

		// tab "Proveedor de Agregación de Rutas"
		JPanel panel2 = new JPanel();
		tabs.addTab("PA Router", panel2);
		//cNode paNode = coreNode.getConnectedPANodes().get(0);
		cNode paNode = coreNode.getConnectedCoreNodes().get(0);
```

```
JTable table3 = new JTable(paNode.getNodeGeneralTable());
table3.getTableHeader().setBackground(Color.yellow);
table3.setRowHeight(20);
Dimension dim3 = new Dimension(20,1);
table3.setIntercellSpacing(new Dimension(dim3));
table3.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
table3.getColumnModel().getColumn(0).setPreferredWidth(200);
table3.getColumnModel().getColumn(1).setPreferredWidth(400);
JScrollPane pane3 = new JScrollPane(table3);
panel2.add(table3.getTableHeader());
panel2.add(table3);
panel2.add(pane3);

JTable table4 = new JTable(paNode.getNodeRoutingTable());
table4.getTableHeader().setBackground(Color.yellow);
table4.setRowHeight(20);
Dimension dim4 = new Dimension(20,1);
table4.setIntercellSpacing(new Dimension(dim4));
table4.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
table4.getColumnModel().getColumn(0).setPreferredWidth(60);
table4.getColumnModel().getColumn(1).setPreferredWidth(200);
table4.getColumnModel().getColumn(2).setPreferredWidth(100);
table4.getColumnModel().getColumn(3).setPreferredWidth(60);
JScrollPane pane4 = new JScrollPane(table4);
panel2.add(table4.getTableHeader());
panel2.add(table4);
panel2.add(pane4);

// tab "Provedor de rutas de borde"
final JPanel panel3 = new JPanel();
tabs.addTab("PE Routers", panel3);

//final ArrayList<cNode> peNodes = coreNode.getConnectedPENodes();
final ArrayList<cNode> peNodes = coreNode.getConnectedCoreNodes();
JLabel numPEs = new JLabel("Numero de rutas PE : " + peNodes.size());
final JTextField numPE = new JTextField(10);
String listPEs[] = new String[peNodes.size()];
for (int i=0; i<peNodes.size(); i++) {
        listPEs[i] = peNodes.get(i).name;
}
final JComboBox list = new JComboBox(listPEs);
panel3.add(numPEs);
panel3.add(list);
panel3.add(numPE);
final JTable tablePEGeneral = new JTable();
final JTable tablePERouting = new JTable();

list.addItemListener(new ItemListener() {
        @Override
        public void itemStateChanged(ItemEvent ie) {
                // TODO Auto-generated method stub
                int i = list.getSelectedIndex();

                // eliminar tablas si había un evento anterior mostrándose
                panel3.remove(tablePEGeneral);
                panel3.remove(tablePERouting);

                numPE.setText(peNodes.get(i).name);
                tablePEGeneral.setModel(peNodes.get(i).getNodeGeneralTable());
                tablePEGeneral.getTableHeader().setBackground(Color.yellow);
                tablePEGeneral.setRowHeight(20);
                Dimension dim5 = new Dimension(20,1);
                tablePEGeneral.setIntercellSpacing(new Dimension(dim5));
                tablePEGeneral.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
```

```
                                tablePEGeneral.getColumnModel().getColumn(0).setPreferredWidth(200);
                                tablePEGeneral.getColumnModel().getColumn(1).setPreferredWidth(400);
                                panel3.add(tablePEGeneral.getTableHeader());
                                panel3.add(tablePEGeneral);

        tablePERouting.setModel(peNodes.get(i).getNodeRoutingTable());
                                tablePERouting.getTableHeader().setBackground(Color.yellow);
                                tablePERouting.setRowHeight(20);
                                Dimension dim3 = new Dimension(20,1);
                                tablePERouting.setIntercellSpacing(new Dimension(dim3));
                                tablePERouting.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
                                tablePERouting.getColumnModel().getColumn(0).setPreferredWidth(60);
                                tablePERouting.getColumnModel().getColumn(1).setPreferredWidth(200);
                                tablePERouting.getColumnModel().getColumn(2).setPreferredWidth(100);
                                tablePERouting.getColumnModel().getColumn(3).setPreferredWidth(60);
                                panel3.add(tablePERouting.getTableHeader());
                                panel3.add(tablePERouting);

                        }
                });

                setSize(800, 600);
                setVisible(true);
                setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        }
```

**class cPresentation**

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

public class cPresentation extends JFrame implements MouseListener {

        int nbOver = 0, nbClick = 0;
        String networks[];
        JRadioButton rb[];

        public cPresentation(String networks[]) {

                setTitle("ZigBee Simulación Aplicada");
                this.networks = networks;

                JPanel background = new JPanel();
                JLabel intro = new JLabel("Seleccione una Red:");
                rb = new JRadioButton[networks.length];
                ButtonGroup group = new ButtonGroup();
                JButton ok = new JButton("Continuar");
                ok.addMouseListener(this);
                background.setLayout(new GridLayout(networks.length+2, 1));

                // Setup Frame
                setContentPane(background);
                background.setBackground(Color.lightGray);
                background.add(intro);

                for (int i=0; i<networks.length; i++) {
                        rb[i] = new JRadioButton(networks[i]);
                        group.add(rb[i]);
                        background.add(rb[i]);

                }
                rb[0].setSelected(true);
```

```
                background.add(ok);

                setLocation(100, 100);
                setSize(400, 300);
                setVisible(true);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        }

        /** metodo para la implementación de MouseListener(Click para aceptar)*/
        public void mouseClicked(MouseEvent e) {
                String n = e.getComponent().getName();
                int i;
                for (i=0; i<networks.length; i++)
                        if (rb[i].isSelected() == true)
                                break;
                String optionClicked = rb[i].getText();
//              cNetwork red = new cNetwork(optionClicked);
//              red.displayNetwork();
//              red.drawNetwork();
        }

        public void mouseEntered(MouseEvent e) {
        }

        public void mouseExited(MouseEvent e) {
        }

        public void mousePressed(MouseEvent e) {
        }

        public void mouseReleased(MouseEvent e) {
        }

}
```

**class cSimFrame**

```
import java.awt.*;
import java.awt.event.*;
import java.net.URL;

import javax.swing.*;
public class cSimFrame extends JFrame implements MouseListener{

        cNetwork network;

        JPanel panel;
        JLabel label1, label2, label3, label4, label5, label6, label7, label8, label9, label10;
        final JComboBox<String> methods, scope, scopeDist, pesDist;
        final JTextField mValue, kValue, percentValue, iterValue, vpnsValue;
        JButton ok;

        public cSimFrame (cNetwork network) {

                this.network = network;
                setSize(600, 400);
                setVisible(true);
                setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                setTitle("Simulación: Configuración y Ejecución");
                panel = new JPanel();
                add(panel);
                panel.setLayout(null);
```

```
                label1 = new JLabel("Método: "); label1.setBounds(10, 10, 80, 20); panel.add(label1);

                String methodsList[] = {"MPSS", "HMPSS"}; methods = new JComboBox<String>(methodsList);
methods.setBounds(100, 10, 80, 20); panel.add(methods);
                label2 = new JLabel("Filtro de Bloom: "); label2.setBounds(10, 40, 80, 20); panel.add(label2);
                label3 = new JLabel("        m = "); label3.setBounds(10, 70, 80, 20); panel.add(label3);
                mValue = new JTextField(); mValue.setBounds(100, 70, 40, 20); panel.add(mValue);
mValue.setText("128");
                label4 = new JLabel("        k = "); label4.setBounds(10, 100, 80, 20); panel.add(label4);
                kValue = new JTextField(); kValue.setBounds(100, 100, 40, 20); panel.add(kValue);
kValue.setText("5");
                label5 = new JLabel("Tipo de alcance: "); label5.setBounds(10, 130, 80, 20); panel.add(label5);
                String scopeList[] = {"Variable", "Fixed"}; scope = new JComboBox<String>(scopeList);
scope.setBounds(100, 130, 80, 20); panel.add(scope);
                label6 = new JLabel("% de alcance fijo = "); label6.setBounds(200, 130, 120, 20);
panel.add(label6);
                percentValue = new JTextField(); percentValue.setBounds(310, 130, 40, 20);
panel.add(percentValue);  percentValue.setText("100");
                label7 = new JLabel("Alcance de distribución variable: "); label7.setBounds(10, 160, 160, 20);
panel.add(label7);
                String scopeDistList[] = {"Uniform", "Zipf"}; scopeDist = new JComboBox<String>(scopeDistList);
scopeDist.setBounds(180, 160, 80, 20); panel.add(scopeDist);
                label8 = new JLabel("PEs distribución variable: "); label8.setBounds(10, 190, 160, 20);
panel.add(label8);
                String pesDistList[] = {"Uniform", "Zipf"}; pesDist = new JComboBox<String>(pesDistList);
pesDist.setBounds(180, 190, 80, 20); panel.add(pesDist);

                label9 = new JLabel("Números de VPNs = "); label9.setBounds(10, 250, 120, 20);
panel.add(label9);
                vpnsValue = new JTextField(); vpnsValue.setBounds(140, 250, 40, 20); panel.add(vpnsValue);
vpnsValue.setText("100");
                label10 = new JLabel("Número de iteraciones = "); label10.setBounds(10, 280, 120, 20);
panel.add(label10);
                iterValue = new JTextField(); iterValue.setBounds(140, 280, 40, 20); panel.add(iterValue);
iterValue.setText("10");

                ok = new JButton("Ir!");
                ok.setBounds(200, 300, 60, 40); panel.add(ok);
                ok.addMouseListener(this);
        }

        /** método para implementación del MouseListener */
        public void mouseClicked(MouseEvent e) {
                if (e.getComponent() == ok) {
                        int numGroups = Integer.parseInt(vpnsValue.getText());
                        int numIter = Integer.parseInt(iterValue.getText());
                        String method = (String)methods.getSelectedItem();
                        String scopeType = scope.getSelectedItem().toString().toLowerCase();
                        int percent = Integer.parseInt(percentValue.getText());
                        String scopeDistribution = scopeDist.getSelectedItem().toString().toLowerCase();
                        String pesDistribution = pesDist.getSelectedItem().toString().toLowerCase();
                        int m = Integer.parseInt(mValue.getText());
                        int k = Integer.parseInt(kValue.getText());
                        //new cSimulationGroup(network, numGroups, numIter, method, scopeType, percent,
scopeDistribution, pesDistribution, m, k, 512);
                        new cSimulationGroup(network,numGroups, numIter, method,scopeType, percent,
scopeDistribution, m, k);
                }

        }

        public void mouseEntered(MouseEvent e) {
        }
```

```
	public void mouseExited(MouseEvent e) {
	}

	public void mousePressed(MouseEvent e) {
	}

	public void mouseReleased(MouseEvent e) {
	}

}
```

**class cSimulation**

```
import java.util.ArrayList;
import java.util.Queue;

public class cSimulation {

	cNetwork network;
	int numGroups;
	public ArrayList<cMulticastGroup> groups;

	public int[][] groupsSamples;
	double[][] avgPerVPN;
	double[] totalAvg;

	int m;
	int k;
	int multiple;		// multiple, en caso de DS-MPSS

	public cSimulation() {

	}

	public cSimulation(cNetwork network, int numGroups) {

		this.network = network;
		this.numGroups = numGroups;
		this.groups = new ArrayList<cMulticastGroup>(0);	// con esta instrucción, la lista de grupos se inicializa
		groupsSamples = new int[numGroups][network.nodes.size()];

	}

	public void genGroupSamples(String scope_type, double scope_percent, String scope_dist) {

		int numNodesInvolved[];
		double U[] = new double[numGroups];

		// Aqui encontramos el numero de nodos involucrados en cada grupo, de acuerdo con el tipo y densidad de la muestra
		if (scope_type.equals("fixed"))  {
			numNodesInvolved = new int[1];
			numNodesInvolved[0] = (int)Math.ceil(scope_percent*network.nodes.size());
		}
		else {// (scope_type == "variable")
			if (scope_dist.equals("uniform"))
				U = SimTools.generateUniformNumbers(numGroups); // matriz de números aleatorios de distribución uniforme entre 0 y 1
			else // if (dist_type == "pareto")
				U = SimTools.generateParetoNumbers(numGroups);
			numNodesInvolved = SimTools.scalarMultiply(network.nodes.size(), U);
```

```
                }

                int x = 0;
                int[] aux;
                for (int i=0; i<numGroups; i++) {
                        if (scope_type.equals("fixed"))
                                x = numNodesInvolved[0];
                        else if (scope_type.equals("variable"))
                                x = numNodesInvolved[i];
                        aux = SimTools.one_comb(network.nodes.size(), x); // selecciona los nodos que van a
abarcar este grupo
                        // aqui se crea la matriz de grupos x nodos
                        for (int j=0; j<aux.length; j++)
                                groupsSamples[i][aux[j]-1] = 1;
                }

        }

        public void displayGroups() {
                /* Metodo para mostrar la matriz de VPNsSamples */
                for (int i=0; i<groupsSamples.length; i++) {
                        System.out.print("Grupo " + (i+1) + ": ");
                        for (int j=0; j<groupsSamples[i].length; j++) {
                                if (groupsSamples[i][j] == 1)
                                        System.out.print((j+1) + ", ");
                        }
                        System.out.println();
                }
        }

        public void attachGroupsToNetwork() {
                /* Metodo para fijar los grupos creados en la red */
                for (int i=0; i<groupsSamples.length; i++) {
                        cMulticastGroup group = new cMulticastGroup(i+1, groupsSamples[i], network);    //
yes! "red" se comporta como un parametro por referencia
                        groups.add(group);
                }
        }

        public void genRandomNodeIDs(int m, int k) {
                this.m = m;
                this.k = k;
                // procedimiento para asignar un  "casi-unico" ID para cada nodo
                for (int i=0; i<network.nodes.size(); i++) {
                        network.nodes.get(i).mpssNodeID = SimTools.generateRandomID(m,k);    // generación
de ID aleatorios, por ejemplo: m=256 y k=5
//                      System.out.print(network.links.get(i).nodeSource.name + "-->" +
network.links.get(i).nodeDest.name + ": ");
//                      MyUtils.displayBits(network.links.get(i).mpssLinkID);
                }
        }

        public void run() {
                // Simulación, cual es origen-ruta, sin estado, y utilización de filtro de Bloom

                this.avgPerVPN = new double[numGroups][7];
                this.totalAvg = new double[7];
                cNode sourceNode;
                int ttl;

                for (int i=0; i<groups.size(); i++) {    // para cada grupo
                //for (int i=1; i<2; i++) {     // para cada grupo
                        cMulticastGroup group = groups.get(i);          // grupo al que sera analizado
```

```
                    int[][] resultsPerSource = new int[group.members.size()][7];     // 2D matriz que
recoge resultados
                    if (group.members.size() > 1) {          // sólo entrega paquetes si hay más de un miembro
                        for (int j=0; j<group.members.size(); j++) {     // para cada nodo origen
miembro del grupo
                        //for (int j=2; j<3; j++) {      // para cada nodo origen miembro del grupo
                            sourceNode = group.members.get(j);// captura el nodo origen
                            int[] pred = network.dijkstra(sourceNode, "hops");       // Encuentra
la matriz pred dijkstra, teniendo en cuenta el numero de saltos
                            // red.displayPredList(pred);           // Visualización de la lista de
pred
                            network.buildLinksTree(pred);        // se construye el árbol
broadcast de toda la red
                            short outterBF[] = network.buildOutterBF(sourceNode);            //
recorre el árbol broadcast y genera los BF resúmenes

                            ArrayList<cLink> linksGroupTree  =
SimTools.buildLinksTree(sourceNode, network, group, pred);
                            MyUtils.displayLinksGroupTree(linksGroupTree, group.groupID);

                            // procedimiento para limpiar todos los childrenGroupBF's de los
nodos
                            for (int a=0; a<network.nodes.size(); a++)
                                network.nodes.get(a).childrenGroupBF = new short[m];
//////////////

                            short childBF[] = group.buildChildBF(sourceNode, linksGroupTree,
m); /////////////////

                            short labelBF[] = group.calculateLabelBF(linksGroupTree);

                            for (int k=0; k<network.nodes.size(); k++) {
                                //System.out.print(network.nodes.get(k).name + "
\n(childrenBF): ");
                                //System.out.print(network.nodes.get(k).name + "
\n(childrenBF): ");
                                MyUtils.displayBits(network.nodes.get(k).childrenBF);
                                //System.out.print("(childrenGroupBF): ");
                                //System.out.print("(childrenGroupBF): ");
                                MyUtils.displayBits(network.nodes.get(k).childrenGroupBF);
                                //System.out.print("(mpssNodeID): ");
                                //System.out.print("(mpssNodeID): ");
                                MyUtils.displayBits(network.nodes.get(k).mpssNodeID);
                            }

                        //          System.out.print("labelBF: ");
        MyUtils.displayBits(labelBF);*/

                            //short innerBF[] = network.buildInnerBF(sourceNode,
linksGroupTree, group);     // recorre el árbol multicast y genera los BF resúmenes

                            // ahora los paquetes con labelBF son lanzados desde el nodo origen
a todo el resto de miembros de PE
                            ttl = SimTools.calculateHeight(sourceNode, network, group, pred);
                            //resultsPerSource[j] = group.launchPacket(sourceNode, 1, outterBF,
innerBF, linksGroupTree, ttl);
                            resultsPerSource[j] = group.launchPacketMPSS(sourceNode, labelBF,
linksGroupTree, ttl);
                            //
        displayResultsPerSource(sourceNode.name, resultsPerSource[j]);

                            avgPerVPN[i][0] = avgPerVPN[i][0] + resultsPerSource[j][0];
                            avgPerVPN[i][1] = avgPerVPN[i][1] + resultsPerSource[j][1];
                            avgPerVPN[i][2] = avgPerVPN[i][2] + resultsPerSource[j][2];
                            avgPerVPN[i][3] = avgPerVPN[i][3] + resultsPerSource[j][3];
```

```
                                        avgPerVPN[i][4] = avgPerVPN[i][4] + resultsPerSource[j][4];
                                        //avgPerVPN[i][5] = avgPerVPN[i][5] +
resultsPerSource[j][5];//nuevo
                                        avgPerVPN[i][6] = avgPerVPN[i][6] + resultsPerSource[j][6];

                                }
                        }
                        // Resultados Promedios
                        avgPerVPN[i][0] = (double) (avgPerVPN[i][0] / group.members.size());
                        avgPerVPN[i][1] = (double) (avgPerVPN[i][1] / group.members.size());
                        avgPerVPN[i][2] = (double) (avgPerVPN[i][2] / group.members.size());
                        avgPerVPN[i][3] = (double) (avgPerVPN[i][3] / group.members.size());
                        avgPerVPN[i][4] = (double) (avgPerVPN[i][4] / group.members.size());
                        //avgPerVPN[i][5] = (double) (avgPerVPN[i][5] / group.members.size());//nuevo
                        avgPerVPN[i][6] = (double) (avgPerVPN[i][6] / group.members.size());

                        displayResultsPerVPN(""+(i+1), avgPerVPN[i]);
                        if (i%100 == 0)
                                System.out.print("Group " + (i+1) + "   ");

                        totalAvg[0] = totalAvg[0] + avgPerVPN[i][0];
                        totalAvg[1] = totalAvg[1] + avgPerVPN[i][1];
                        totalAvg[2] = totalAvg[2] + avgPerVPN[i][2];
                        totalAvg[3] = totalAvg[3] + avgPerVPN[i][3];
                        totalAvg[4] = totalAvg[4] + avgPerVPN[i][4];
                        //totalAvg[5] = totalAvg[5] + avgPerVPN[i][5];//nuevo
                        totalAvg[6] = totalAvg[6] + avgPerVPN[i][6];

                }

        }

        public void displayResultsPerSource (String title, int results[]) {
                /* Método para visualizar los resultados:
                 *                      0: Promedio de paquetes perdidos por radiación
                 *                      1: Promedio de bucles detectados y cortados
                 *                      2: Promedio de ancho de banda desperdiciado
                 *                      3: Promedio de ancho de banda utilizado
                 *                      4: Promedio de errores(Si Cualquier) = Enlace utilizado - BW usado
                 */
                System.out.println("\nRESULTADOS de " + title);
                System.out.println("Paquetes perdidos por radiación:\t" + results[0]);
                System.out.println("Bucles detectados:\t" + results[1]);
                System.out.println("Ancho de banda desperdiciado:\t\t" + results[2]);
                System.out.println("Ancho de banda utilizado:\t\t" + results[3]);
                System.out.println("Errores:\t\t\t" + results[4]);
                //Avg. size of initial label stack
                System.out.println("Promedio inicial del tamaño de la etiqueta de la pila:\t" + results[5]);
                System.out.println("Promedio de bytes de gastos generales de la cabecera :\t" + results[6]);

        }

        public void displayResultsPerVPN (String title, double results[]) {
                /* Método para visualizar los resultados:
                 *                      0: Promedio de paquetes perdidos por radiación
                 *                      1: Promedio de bucles detectados y cortados
                 *                      2: Promedio de ancho de banda desperdiciado
                 *                      3: Promedio de ancho de banda utilizado
                 *                      4: Promedio de errores (if any) = Enlaces utilizados - BW usado
                 */
                System.out.println("\nRESULTADOS de Grupo " + title);
                System.out.println("Promedio de paquetes desperdiciados por radiación:\t\t" + results[0]);
                System.out.println("Promedio de bucles detectados:\t" + results[1]);
                System.out.println("Promedio de ancho de banda desperdiciado:\t\t" + results[2]);
```

```
                System.out.println("Promedio de ancho de banda utilizado:\t\t" + results[3]);
                System.out.println("Promedio de errores:\t\t\t\t" + results[4]);
                System.out.println("Promedio del tamaño inicial de la etiqueta de la pila:\t" + results[5]);
                System.out.println("Promedio de bytes de gastos generales de la cabecera:\t" + results[6]);
        }

        public void displayResultsPerIteration (int numIter) {
                /* Método para visualizar la matriz de resultados:
                 *              0: Paquetes perdidos por radiación
                 *              1: Bucles detectados y reducidos
                 *              2: Ancho de banda desperdiciado
                 *              3: Ancho de banda utilizaso
                 *              4: Errores (if any) = Enlaces utilizados - BW utilizado
                 */
                System.out.println("\nRESULTADOS DE ITERACIONES " + (numIter+1) +
"\n=====================");
                System.out.println("Red: " + network.name);
//              System.out.println("Number of core nodes: " + network.numberOfCoreNodes);
//              System.out.println("Number of aggregation nodes: " + network.numberOfPANodes);
//              System.out.println("Number of PE nodes: " + network.numberOfPENodes);
//              System.out.println("Total number of nodes: " + network.totalNumberOfNodes);
//              System.out.println("Total number of links: " + network.totalNumberOfLinks);
//              System.out.println("Number of VPNs: " + vpns.size());
//              System.out.println("m = " + m + " ,  k = " + k);
                System.out.println("        Paquetes perdidos por radiación:\t" + totalAvg[0]);
                System.out.println("        Bucles detectados:\t" + totalAvg[1]);
                System.out.println("        Ancho de banda desperdiciado:\t" + totalAvg[2]);
                System.out.println("        Ancho de banda utilizado:\t" + totalAvg[3]);
                System.out.println("        Errores:\t" + totalAvg[4]);
                System.out.println("        Promedio del tamaño inicial de la etiqueta de la pila:\t" +
totalAvg[5]);
                System.out.println("        Promedio de bytes de gastos generales de la cabecera:\t" +
totalAvg[6]);

        }

}
```

**class cSimulationGroup**

```
public class cSimulationGroup {

        cNetwork network;
        String method, scopeType, scopeDist, pesDist;
        int numIter, numGroups, m, k, multiple;
        double totalAvg[] = new double[7];
        double scopePercent;

        public cSimulationGroup (cNetwork network, int numGroups, int iter, String method, String scopeType,
                                                                        double scopePercent, String scopeDist,
int m, int k) {

                double total[] = new double[7];
                this.network = network;
                this.numGroups = numGroups;
                this.numIter = iter;
                this.method = method;
                this.scopeType = scopeType;
                this.scopePercent = scopePercent;
                this.scopeDist = scopeDist;
                this.m = m;
                this.k = k;
                this.network.createTableNeighbors();
```

```
		cSimulation[] sim = new cSimulation[numIter];		// crea la matriz de simulaciones individuales
		for (int i=0; i<sim.length; i++) {
			sim[i] = new cSimulation(network, numGroups);

			if (scopeType.equals("fixed"))
				sim[i].genGroupSamples(scopeType, scopePercent, scopeDist);
			else
				sim[i].genGroupSamples(scopeType, 1, scopeDist);

/*			sim[i].groupsSamples[0][0] = 0;
			sim[i].groupsSamples[0][1] = 1;
			sim[i].groupsSamples[0][2] = 0;
			sim[i].groupsSamples[0][3] = 1;
			sim[i].groupsSamples[0][4] = 1;
			sim[i].groupsSamples[0][5] = 0;
			sim[i].groupsSamples[0][6] = 0;
			sim[i].groupsSamples[0][7] = 0;

			sim[i].groupsSamples[1][0] = 1;
			sim[i].groupsSamples[1][1] = 1;
			sim[i].groupsSamples[1][2] = 1;
			sim[i].groupsSamples[1][3] = 0;
			sim[i].groupsSamples[1][4] = 0;
			sim[i].groupsSamples[1][5] = 0;
			sim[i].groupsSamples[1][6] = 1;
			sim[i].groupsSamples[1][7] = 1;

			sim[i].groupsSamples[2][0] = 0;
			sim[i].groupsSamples[2][1] = 0;
			sim[i].groupsSamples[2][2] = 1;
			sim[i].groupsSamples[2][3] = 0;
			sim[i].groupsSamples[2][4] = 1;
			sim[i].groupsSamples[2][5] = 1;
			sim[i].groupsSamples[2][6] = 0;
			sim[i].groupsSamples[2][7] = 1;

			sim[i].groupsSamples[3][0] = 1;
			sim[i].groupsSamples[3][1] = 0;
			sim[i].groupsSamples[3][2] = 0;
			sim[i].groupsSamples[3][3] = 1;
			sim[i].groupsSamples[3][4] = 1;
			sim[i].groupsSamples[3][5] = 1;
			sim[i].groupsSamples[3][6] = 0;
			sim[i].groupsSamples[3][7] = 0;

			sim[i].groupsSamples[4][0] = 0;
			sim[i].groupsSamples[4][1] = 1;
			sim[i].groupsSamples[4][2] = 1;
			sim[i].groupsSamples[4][3] = 1;
			sim[i].groupsSamples[4][4] = 1;
			sim[i].groupsSamples[4][5] = 0;
			sim[i].groupsSamples[4][6] = 0;
			sim[i].groupsSamples[4][7] = 0;
*/
			sim[i].attachGroupsToNetwork();	// procedimiento para crear y conectar los grupos generados aleatoriamente a la red (random-generated)

			if (method.equals("WAHN-BF")) {// (**M**)El siguiente punto estado activo para la generación aleatoria de IDs
				sim[i].genRandomNodeIDs(m, k);	// procedimiento para asignar un "casi-unico" ID para cada nodo
				sim[i].run(); // ejecucion de la simulación
			}
```

```
                        sim[i].displayResultsPerIteration(i);

                        for (int j=0; j<total.length; j++)
                                total[j] = total[j] + sim[i].totalAvg[j];

                }

                for (int j=0; j<totalAvg.length; j++)
                        totalAvg[j] = total[j] / numIter;

        }

        public void displayTotalResults () {
                /* Método para visualizar los resultados de la matriz:
                 *                      0: Falsos positivos
                 *                      1: Bucles detectados y reducidos
                 *                      2: Ancho de banda desperdiciado
                 *                      3: Ancho de banda utilizado
                 *                      4: Errores (if any) = Enlaces utilizados - BW utilizado
                 */
                System.out.println("\nTOTAL RESULTADOS\n=============");
                System.out.println("MÉTODO: " + method);
                System.out.println("Red: " + network.name);
                System.out.println("Número de nodos núcleo: " + network.numberOfWSNNodes);
//              System.out.println("Número de nodos agregados: " + network.numberOfPANodes);
//              System.out.println("Número de nodos PE: " + network.numberOfPENodes);
                System.out.println("Total número de nodos: " + network.numberOfWSNNodes);
                System.out.println("Total número de enlaces (virtual): " + network.links.size());
                System.out.println("Método: " + method);
                System.out.println("Numero de grupos: " + numGroups);
                System.out.println("Tipo de alcance: " + scopeType);
                System.out.println("Distribución de alcance: " + scopeDist);
                System.out.println("Porcentaje de alcance: " + scopePercent);
//              System.out.println("PEs distribution: " + pesDist);
                System.out.println("Numero de iteraciones: " + numIter);
                System.out.println("m = " + m + " ,  k = " + k);
                System.out.println("        Paquetes perdidos por radiación:\t" + totalAvg[0]);
                System.out.println("        Bucles detectados:\t" + totalAvg[1]);
                System.out.println("        Ancho de banda desperdiciado:\t" + totalAvg[2]);
                System.out.println("        Ancho de banda utilizado:\t" + totalAvg[3]);
                System.out.println("        Errores:\t" + totalAvg[4]);
                System.out.println("        Tamaño de la etiqueta de la pila:\t" + totalAvg[5]);
                System.out.println("        Gastos generales de la cabecera (bytes):\t" + totalAvg[6]);

        }

}
```

**class cVPN**

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Stack;

public class cVPN {

        // identificación de atributos
        int vpnID;                                              // ID de la VPN

        // atributos con respecto a la distribución de la VPN
        int destNodes[][];                      // 2d-matriz de CRs y PEs involucrados
        ArrayList<cNode> destPEs; // lista de PEs que unen miembros VPN
```

```
	cNetwork network;

	// atributos para la agregación
	short[] vpnMPFS_ID;				// ID aleatoria de la VPN, para la tecnica MPFS
	ArrayList<short[]> bloomFilter1;	// lista de bloomFilters1 (hay un nodo origen por VPN)

	public cVPN(int vpnID, cNetwork network) {
		/* Constructor si una matriz de PEs el indice no es proporcionado */
		this.vpnID = vpnID;
		this.network = network;
		destPEs = new ArrayList<cNode>(0);

	}

	public cVPN(int vpnID, int indexPEs[][], cNetwork network) {
		/* Construir si una matriz de PEs el indice son proporcionadas */

		this.vpnID = vpnID;
		this.network = network;
		int nPEs = 0;
		destPEs = new ArrayList<cNode>(0);		// initialización de ña lista de PE nodos que son miembros de esta VPN
		bloomFilter1 = new ArrayList<short[]>(0);	// inicialización de la lista de bloomFilters1 (hay un nodo fuente por PE)
		//		indexPEs is m x n, where n (cols) = number of core nodes, m (rows) = max number of PEs per core node

		destNodes = new int[indexPEs[0].length][];		// creación de columnas
		for (int i=0; i<network.PEs.length; i++) {
			nPEs = 0;
			for (int j=0; j<indexPEs.length; j++) {
				if (indexPEs[j][i] == 1)
					nPEs++;
				destNodes[i] = new int[nPEs+1];			// Creación de filas
			}
		}

		for (int i=0; i<network.PEs.length; i++) {
			destNodes[i][0] = network.nodes.get(i).nodeID;
			nPEs = 0;
			for (int j=0; j<indexPEs.length; j++)
				if (indexPEs[j][i] == 1) {

					bloomFilter1.add(null);	// La lista de bloomfilters1 crece con valores nulos

					nPEs++;
				}
		}

	}

	public short[] calculateLabelBF(ArrayList <cLink> linksTree) {
		/* Procedimiento de la creacion de la etiqueta de filtro de Bloom, al unirse (con OR) los MPSS enlacesID's del linksTree */

		int m = linksTree.get(0).mpssLinkID.length;
		short labelBF[] = new short[m];	// es la etiqueta del filtro de Bloom que va hacer encontrado
		for (int i=0; i<linksTree.size(); i++)	// para cada enlace del linksTree de la VPN
			labelBF = MyUtils.OR(labelBF, linksTree.get(i).mpssLinkID);	// debido a la propiedad asociativa la operación OR, labelBF recopila los resultados en cada iteración
		return labelBF;

	}
```

```
	//nuevo cambio
	public ArrayList<Short[]> calculateLabelBFStack_sensor(ArrayList <cLink> linksTree, cNode sourceNode)
{
		/* Procedimiento de la creación de la etiqueta del filtro de Bloom, al unirse (con OR) los MPSS
linkID's del linksTree */

		ArrayList<Short[]> labelBFStack = new ArrayList<Short[]>();
		Queue<cNode> pila = new LinkedList<cNode>();
		Queue<Integer> deeps = new LinkedList<Integer>();
		int m = linksTree.get(0).mpssLinkID.length;

		// ahora se tiene que construir la etiqueta de la pila yendo desde el nodo origen a los nodos
miembros PE
		int deep = 0;
		deeps.add(deep);
		cNode pivot = sourceNode;
		pila.add(pivot);
		short labelBF[] = new short[m];	// esta es la etiqueta del filtro de Bloom que va a ser
encontrado en cada nivel

		do {
			pivot = pila.poll();
			if (deep < deeps.element()) {		// si ahora se esta a un nivel mas profundo
				deep = deeps.poll();
				labelBFStack.add(MyUtils.shortToShort(labelBF)); // esta etiqueta de filtro de
Bloom que va a ser encontrado para este nivel
				labelBF = new short[m];	// limpia el labelBF
			}
			else
				deeps.remove();
			for (int i=0; i<pivot.outLinks.size(); i++) {
				cLink link = pivot.outLinks.get(i);
				cNode node = pila.element();//Verificar
				if (linksTree.contains(link)) {
					labelBF = MyUtils.OR(labelBF, node.mpssNodeID);//verificar
					pila.add(link.nodeDest);
					deeps.add(deep+1);		// el nivel corresponsal se presiona sobre
la profundidad de la pila
				}
			}
		}
		while (!pila.isEmpty());	// aunque todavia haya nodos en la pila

		return labelBFStack;

	}

	public int[] launchPacketMPSS_ttl(cNode sourceNode, short labelBF[], ArrayList<cLink> linksTree, int
ttlValue) {
		/* Ahora el paquete con labelBF se lanza desde el nodo origen a los nodos destinos PE
		 * Calcula el ancho de banda desperdiciado que deberia haber
		 */
		/* Este método no recursivo realiza una busqueda de la primera amplitud (BFS) del árbol */

		ArrayList<cNode> cola = new ArrayList<cNode>(0);	//		crea la cola
		ArrayList<Boolean> colaPath = new ArrayList<Boolean>(0);
		ArrayList<Integer> colaTTL = new ArrayList<Integer>(0);
		ArrayList<Integer> colaPredIndex = new ArrayList<Integer>(0);
		cNode pivot, previousNode;
		boolean rightPath, loop;
		int ttl, ttlPath, iNextPivot = 0, nColas = 1, auxColas, temp, endPackets = 0;
		int m = labelBF.length;
```

```
                cola.add(sourceNode);
                colaPredIndex.add(0);
                colaPath.add(true);
                colaTTL.add(ttlValue);
                pivot = new cNode();
                int bwUsed = 0;
                int falsePositives = 0;
                int bwWasted = 0;
                int loopsDetected = 0;

                while (nColas > iNextPivot) {

                        pivot = cola.get(iNextPivot);
                        previousNode = cola.get(colaPredIndex.get(iNextPivot));
                        rightPath = colaPath.get(iNextPivot);
                        ttlPath = colaTTL.get(iNextPivot);
                        auxColas = iNextPivot;
                        //                      System.out.print("\n" + previousNode.name + " --> " +
pivot.name + " --> " );

                        for (int i=0; i<pivot.outLinks.size(); i++) {
                                cLink auxLink = pivot.outLinks.get(i);          // auxLink es el enlace a probar
ahora
                                ttl = ttlPath;
                                //                              System.out.print("\n
        " + auxLink.nodeDest.name + "   ttl: " + ttl);
                                if (previousNode != auxLink.nodeDest) {         // esto es para evitar volver al
nodo predecesor
                                        if (ttlPath > 0)        { // evita retornar al nodo anterior o bucle infinito
(a pesar que no pasa por el BF)
                                                if (MyUtils.BloomFilter(labelBF, auxLink.mpssLinkID)) { // si
labelBF reemplaza al BF de este enlace
                                                        if (linksTree.contains(auxLink)) {      // el enlace es
parte del árbol
                                                                if (rightPath == true) {        // si el camino
es el correcto
                                                                        bwUsed++;
                                                                        //System.out.print(" OK");
                                                                        if (auxLink.nodeDest.type ==
"PE") {
                                                                                endPackets++;
                                                                                //System.out.print("
Leaf!"); // si se alcanza un nodo hoja
                                                                        }
                                                                        else {
        cola.add(auxLink.nodeDest);
                                                                                colaTTL.add(ttlPath-1);
                                                                                colaPath.add(true);
        colaPredIndex.add(auxColas);
                                                                                nColas++;
                                                                        }
                                                                }
                                                                else {  // si el camino no es el correcto
                                                                        bwWasted++;
                                                                        ttl--;
                                                                        //System.out.print(" Paquete
desperdiciado");
                                                                        if (auxLink.nodeDest.type ==
"PE") {
                                                                                //System.out.print("
Hoja!"); // si se ha alcanzado un nodo hoja
                                                                        }
```

```
                                                                    else {

            cola.add(auxLink.nodeDest);
                                                                            colaTTL.add(ttlPath-1);
                                                                            colaPath.add(false);

            colaPredIndex.add(auxColas);
                                                                            nColas++;
                                                                    }
                                                            }
                                                    }
                                                    else    {// el enlace no es parte del árbol (falso
positivo)
                                                            if (rightPath == true) {        // si el camino
es correcto hasta ahora
                                                                    falsePositives++;
                                                                    //System.out.print(" Falso
Positivo");
                                                            }
                                                            else {      // si el camino no es correcto
                                                                    bwWasted++;
                                                                    //System.out.print(" Paquete
desperdiciado");
                                                            }
                                                            ttl--;
                                                            temp = colaPredIndex.get(iNextPivot);
                                                            loop = false;
                                                            while (cola.get(temp) != sourceNode) {
                                                                    if (cola.get(temp) ==
auxLink.nodeDest) {             // esto va a generar un bucle
                                                                            loopsDetected++;  // el
numero de bucles es incrementado y no hace nada
                                                                            //System.out.print("
Bucle detectado");
                                                                            loop = true;
                                                                            break;
                                                                    }
                                                                    else
                                                                            temp =
colaPredIndex.get(temp);
                                                            }
                                                            //if (loop != true) { // esto no va a generar
algun bucle y no se ha logrado un nodo hoja (Por tanto, reducimos los bucles)
                                                            if (auxLink.nodeDest.type == "PE") {
                                                                    //System.out.print(" Leaf!"); //
se a alcanzado un nodo hoja
                                                            }
                                                            else {
                                                                    cola.add(auxLink.nodeDest);
                                                                    colaTTL.add(ttlPath-1);
                                                                    colaPath.add(false);
                                                                    colaPredIndex.add(auxColas);
                                                                    nColas++;
                                                            }
                                                            //}
                                                    }
                                            }
                                    }
                                    else {      // si ttlValue < 0 el recorrido del paquete se corta o
suspende (no deberia suceder,pero esta aqui por si es necesario)
                                            //System.out.println(" TTL out");
                                    }
                            }
                    }
```

```
                    iNextPivot++;

                }

                int results[] = new int[7];
                results[0] = falsePositives;
                results[1] = loopsDetected;
                results[2] = bwWasted;
                results[3] = bwUsed;          // bwUsed = bw useful
                results[4] = ( (this.destPEs.size() - 1) - endPackets) + (linksTree.size() - bwUsed );  //
comprobación de errores, 1: paquetes que no llegaron a destPEs, 2: tree == bwUsed

                results[6] = m * (bwUsed + falsePositives + bwWasted) / 8;        // calculation of header
overhead (in number of bytes): sum of links where packets have gone through

                return results;

        }

}
```

**class MyUtils**

```
import java.util.ArrayList;

public class MyUtils {

        public static Short[] shortToShort (short a[]) {
                /* Este método devuelve una matriz Short[] de una matriz primitiva short[] */
                Short result[] = new Short[a.length];
                for (int i=0; i<result.length; i++)
                        result[i] = new Short(a[i]);
                return result;
        }

        public static short[] shortToShort (Short a[]) {
                /* Este método devuelve una matriz Short[] de una matriz primitiva short[] */
                short result[] = new short[a.length];
                for (int i=0; i<result.length; i++)
                        result[i] = a[i].shortValue();
                return result;
        }

        /* Este método devuelve la operación OR de dos matrices de bits (tipo short)
         * Ambos parametros deben tener la misma longitud */
        public static short[] OR(short a[], short b[]) {

                if (a.length == b.length) {    // verificación de ambos parametros que tengan la misma longitud
                        short result[] = new short[a.length];
                        for (int i=0; i<a.length; i++)
                                result[i] = (short)(a[i] | b[i]);
                        return result;
                }
                else
                        return null;

        }

        /* El método devuelve la operación OR de dos matrices de bits (tipo Short)
         * Ambos parametros deben tener la misma longitud */
        public static Short[] OR(Short a[], Short b[]) {

                if (a.length == b.length) {    // verificación de ambos parametros que tengan la misma longitud
                        Short result[] = new Short[a.length];
```

```
                for (int i=0; i<a.length; i++)
                        result[i] = new Short((short)(a[i] | b[i]));
                return result;
        }
        else
                return null;

}

/* Este método devuelve la operación AND de dos matrices de bits
 * Ambos parametros deben tener la misma longitud */
public static short[] AND(short a[], short b[]) {

        if (a.length == b.length) { // verificación de ambos parametros que tengan la misma longitud
                short result[] = new short[a.length];
                for (int i=0; i<a.length; i++)
                        result[i] = (short)(a[i] & b[i]);
                return result;
        }
        else
                return null;

}

/* Este método devuelve true si a y b (matriz de bits) son iguales*/
public static boolean EQUAL(short a[], short b[]) {

        if (a.length == b.length) { // verificación de ambos parametros que deben tener la misma longitud
                boolean equal = true;
                for (int i=0; i<a.length; i++)
                        if (a[i] != b[i]) {
                                equal = false;
                                break;
                        }
                if (equal == true)
                        return true;
                else
                        return false;
        }
        else
                return false;

}

/* El método calcula, dado un filtro de Bloom, el promedio del factor llenado (p)
 */
public static double fillFactor(short BF[]) {

        int bits_1 = 0;
        for (int i=0; i<BF.length; i++)         // este encuentra bits_1 (El número de 1s en el BF)
                bits_1 = bits_1 + BF[i];

        double result = ((double)bits_1)/BF.length;
        return result;

}

/* Este método convierte un largo Filtro de Bloom de longitud M a un iBF de longitud m
 */
public static short[] convertBF(short[] longBF, int m) {

        int M = longBF.length, x;

        short[] iBF = new short[m];
```

```
		for (int i=0; i<m; i++) {
			x = (int)Math.ceil((double)M/m);
			for (int j=0; j<x; j++)
				if ( (j*m+i) < M )   // si excede los limites de longBF
					iBF[i] = (short) ( iBF[i] | longBF[j*m + i] );
				else
					iBF[i] = (short) ( iBF[i] | 0 );
		}
		return iBF;

	}

	public static boolean BloomFilter(short a[], short filter[]) {

		short[] result = AND(a, filter);
		if (EQUAL(result,filter))
			return true;
		else
			return false;

	}

	public static void displayBits(short[] arrayOfBits) {

		for (int i=0; i<arrayOfBits.length; i++) {
			//(MODIFICADO)System.out.print(arrayOfBits[i] + "");
		}
		//(MODIFICADO)System.out.println();

	}

	public static void displayArraysOfBits(ArrayList<Short[]> list) {

		for (int i=0; i<list.size(); i++) {
			displayBits(shortToShort(list.get(i)));
		}
		System.out.println();

	}

	public static void displayNumbers(int[] arrayOfNumbers) {

		for (int i=0; i<arrayOfNumbers.length; i++) {
			System.out.print(arrayOfNumbers[i] + " ");
		}
		System.out.println();

	}

	public static void displayLinksGroupTree(ArrayList<cLink> linksGroupTree, int numGroup) {
		//(MODIFICADO)System.out.println("LinksGroupTree " + numGroup);
		for (int i=0; i<linksGroupTree.size(); i++) {
			//(MODIFICADO)System.out.println(linksGroupTree.get(i).nodeSource.name + " --> " +
linksGroupTree.get(i).nodeDest.name);
		}
	}

}
```

**class principal**

```
import java.util.ArrayList;
```

```
import org.w3c.dom.Node;

public class principal {

	public static final String nodes = null;

	public static void main(String[] args) {

		new cPresentation(new
String[]{"WArequipa","WPeru","Test","Prueba","Abilene","NSFNET","COST-266"});

		// creación de la red
		//cNetwork network = new cNetwork("WArequipa", 8, 86);
		cNetwork network = new cNetwork("WArequipa", 8, 100);
		cSimulationGroup groupSim = new cSimulationGroup(network, 5, 2, "WAHN-BF", "variable", 1,
"uniform", 8, 1);
		//cSimulationGroup groupSim = new cSimulationGroup(network, 5, 2, "WAHN-BF", "fixed", 0.3,
"uniform", 8, 1);
		//cSimulationGroup groupSim = new cSimulationGroup(network, 50, 5, "WAHN-BF", "fixed", 0.5,
"uniform", 8, 1);
		//cSimulationGroup groupSim = new cSimulationGroup(network, #Grupos, Iteraciones, "WAHN-
BF", "fixed", 0.5 , "uniform", m, k);
		//WArequipa
/*		network.nodes.get(0).mpssNodeID = new short[8];
		network.nodes.get(0).mpssNodeID[0] = 1;
		network.nodes.get(0).mpssNodeID[1] = 0;
		network.nodes.get(0).mpssNodeID[2] = 0;
		network.nodes.get(0).mpssNodeID[3] = 0;
		network.nodes.get(0).mpssNodeID[4] = 0;
		network.nodes.get(0).mpssNodeID[5] = 0;
		network.nodes.get(0).mpssNodeID[6] = 0;
		network.nodes.get(0).mpssNodeID[7] = 0;

		network.nodes.get(1).mpssNodeID = new short[8];
		network.nodes.get(1).mpssNodeID[0] = 0;
		network.nodes.get(1).mpssNodeID[1] = 1;
		network.nodes.get(1).mpssNodeID[2] = 0;
		network.nodes.get(1).mpssNodeID[3] = 0;
		network.nodes.get(1).mpssNodeID[4] = 0;
		network.nodes.get(1).mpssNodeID[5] = 0;
		network.nodes.get(1).mpssNodeID[6] = 0;
		network.nodes.get(1).mpssNodeID[7] = 0;

		network.nodes.get(2).mpssNodeID = new short[8];
		network.nodes.get(2).mpssNodeID[0] = 0;
		network.nodes.get(2).mpssNodeID[1] = 0;
		network.nodes.get(2).mpssNodeID[2] = 1;
		network.nodes.get(2).mpssNodeID[3] = 0;
		network.nodes.get(2).mpssNodeID[4] = 0;
		network.nodes.get(2).mpssNodeID[5] = 0;
		network.nodes.get(2).mpssNodeID[6] = 0;
		network.nodes.get(2).mpssNodeID[7] = 0;

		network.nodes.get(3).mpssNodeID = new short[8];
		network.nodes.get(3).mpssNodeID[0] = 1;
		network.nodes.get(3).mpssNodeID[1] = 0;
		network.nodes.get(3).mpssNodeID[2] = 0;
		network.nodes.get(3).mpssNodeID[3] = 0;
		network.nodes.get(3).mpssNodeID[4] = 0;
		network.nodes.get(3).mpssNodeID[5] = 0;
		network.nodes.get(3).mpssNodeID[6] = 0;
		network.nodes.get(3).mpssNodeID[7] = 0;

		network.nodes.get(4).mpssNodeID = new short[8];
```

```
                    network.nodes.get(4).mpssNodeID[0] = 0;
                    network.nodes.get(4).mpssNodeID[1] = 0;
                    network.nodes.get(4).mpssNodeID[2] = 0;
                    network.nodes.get(4).mpssNodeID[3] = 0;
                    network.nodes.get(4).mpssNodeID[4] = 0;
                    network.nodes.get(4).mpssNodeID[5] = 0;
                    network.nodes.get(4).mpssNodeID[6] = 0;
                    network.nodes.get(4).mpssNodeID[7] = 1;

                    network.nodes.get(5).mpssNodeID = new short[8];
                    network.nodes.get(5).mpssNodeID[0] = 0;
                    network.nodes.get(5).mpssNodeID[1] = 0;
                    network.nodes.get(5).mpssNodeID[2] = 0;
                    network.nodes.get(5).mpssNodeID[3] = 1;
                    network.nodes.get(5).mpssNodeID[4] = 0;
                    network.nodes.get(5).mpssNodeID[5] = 0;
                    network.nodes.get(5).mpssNodeID[6] = 0;
                    network.nodes.get(5).mpssNodeID[7] = 0;

                    network.nodes.get(6).mpssNodeID = new short[8];
                    network.nodes.get(6).mpssNodeID[0] = 0;
                    network.nodes.get(6).mpssNodeID[1] = 0;
                    network.nodes.get(6).mpssNodeID[2] = 0;
                    network.nodes.get(6).mpssNodeID[3] = 0;
                    network.nodes.get(6).mpssNodeID[4] = 1;
                    network.nodes.get(6).mpssNodeID[5] = 0;
                    network.nodes.get(6).mpssNodeID[6] = 0;
                    network.nodes.get(6).mpssNodeID[7] = 0;

                    network.nodes.get(7).mpssNodeID = new short[8];
                    network.nodes.get(7).mpssNodeID[0] = 0;
                    network.nodes.get(7).mpssNodeID[1] = 0;
                    network.nodes.get(7).mpssNodeID[2] = 0;
                    network.nodes.get(7).mpssNodeID[3] = 0;
                    network.nodes.get(7).mpssNodeID[4] = 0;
                    network.nodes.get(7).mpssNodeID[5] = 0;
                    network.nodes.get(7).mpssNodeID[6] = 1;
                    network.nodes.get(7).mpssNodeID[7] = 0;
*/

//                  cSimulationGroup groupSim = new cSimulationGroup(network, 5, 1, "WAHN-BF", "variable", 1 ,
"uniform", 8, 1);

                    //cSimulationGroup groupSim = new cSimulationGroup(network, 5, 10, "WAHN-BF", "fixed", 0.25
, "uniform", 4, 1);

                    groupSim.displayTotalResults();

          }

}
```

**class SimTools**

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;

public class SimTools {

          public static boolean isNeighbor(double xc, double yc, double xp, double yp, double r) {
                    /* El método devuelve true si el nodo (xp,yp) esta dentro del circulo los puntos (xc,yc)
                     *
                     */
```

```
		if (Math.sqrt(Math.pow((xc-xp),2) + Math.pow((yc-yp),2)) <= r)
			return true;
		else
			return false;

	}

	public static int distance(double xc, double yc, double xp, double yp) {
		/* Este método devuelve la distancia entre dos puntos
		 *
		 */
		float d = (float)Math.sqrt(Math.pow((xc-xp),2) + Math.pow((yc-yp),2));
		return Math.round(d);
	}

	public static int max (int[] values) {
		/* Este método devuelve el valor maximo de una matriz */
		int max = (int)Double.MIN_VALUE;
		for (int i=0; i<values.length; i++) {
			if (values[i] > max)
				max = values[i];
		}
		return max;
	}

	public static short[] generateRandomID(int m, int k) {
		/* Procedimiento para generar "casi unico" nodo ID aleatorio
		 *			m: número de bits del ID
		 *			k: numero de 1's (tienen que ser muy pocos)
		 * Hay m!/[k!(m-k)]!		combinaciones diferentes
		 * Por ejemplo: m=256, k=5, da 8,809'549,056 posibles IDs */

		short randomID[]  = new short[m];  // crea la matriz para contruir el número binario
		double prob0 = ((double)(m-k))/m;		// probabilidad para un 0 para que aparesca
		Random randomGenerator = new Random();
		double r;
		int sum1s = 0;	// Esta variable controla el número de 1's (si hay mas o menos que k 1s tiene
que generar de nuevo una matriz binaria)

		while (sum1s != k)		{		// el numero binario es correcto(OK) solo si num1s = k
			sum1s = 0;		// sum1s es iniciado nuevamente
			for (int i=0; i<m; i++) {
				r = randomGenerator.nextDouble();  // da un número aleatorio entre 0.0 y 1.0
				if (r <= prob0)	// si 0 < r < prob0
					randomID[i] = 0;
				else {
					randomID[i] = 1;
					sum1s++;
				}
			}
		}
		return randomID;

	}

	public static ArrayList<cLink> buildLinksTree(cNode sourceNode, cNetwork network, cMulticastGroup
group, int[] pred) {
		/* El método "construye" el árbol, mediante la adición de enlaces */
		ArrayList <cLink> linksTree = new ArrayList<cLink>(0);
		for (int n=0; n<group.members.size(); n++) {  // para cada nodo destino del grupo
			cNode auxNode = group.members.get(n);
			while (auxNode != sourceNode) {	// itera desde el nodo hasta conseguir el nodo
origen
```

```
                                        int iAuxNode = auxNode.nodeID-1;   // iAuxNode: posición del nodo en la lista
de nodos
                                        cNode predNode = network.nodes.get(pred[iAuxNode]);          // da el
pred[iAuxNode], devuelve el predNode
                                        cLink predAuxLink = auxNode.getLinkFrom(predNode);          // da el nodo
predecesor, este devuelve los enlace
                                        if (!linksTree.contains(predAuxLink))          // agrega el enlace para el
linksTree
                                                linksTree.add(predAuxLink);
                                        auxNode = predNode;         // ahora el nuevo auxNode es el predNode
                                }
                        }
                        return linksTree;

        }

        public static int calculateHeight(cNode sourceNode, cNetwork network, cMulticastGroup group, int[]
pred) {
                        /* El método calcula la maxima altura(h) del árbol (numero de enlaces desde el destino PE más
lejos al nodo origen)) */
                        int h = 0;
                        int dist;
                        for (int n=0; n<group.members.size(); n++) {  // para cada nodo destino PE del VPN
                                dist = SimTools.distToSource(sourceNode, network, pred, group.members.get(n));
                                if (dist > h)
                                        h = dist;
                        }
        return h;

        }

        public static int distToSource (cNode sourceNode, cNetwork network, int[] pred, cNode destNode) {
                        /* El método calcula la distancia en en saltos desde un dado nodo PE (hoja) a un nodo origen */
                        int distance = 0;
                        cNode auxNode = destNode;
                        while (auxNode != sourceNode) {     // itera desde el nodo PE hasta que se consigue el nodo
origen
                                int iAuxNode = auxNode.nodeID-1;   // iAuxNode: posición del nodo en la lista de nodos
                                cNode predNode = network.nodes.get(pred[iAuxNode]);          // dado el
pred[iAuxNode], retorna el predNode
                                auxNode = predNode;         // ahora el nuevo auxNode es el predNode
                                distance++;
                        }
                        return distance;

        }

        public static double[] generateUniformNumbers(int n) {
                        /* Este método devuelve n-array de numeros variables aleatorios uniforme entre 0.0 y 1.0 */
                        double values[] = new double[n];
                        Random randomGenerator = new Random();
                        for (int i=0; i<n; i++)
                                values[i] = randomGenerator.nextDouble();    // dado un número aleatorio entre 0.0 y
1.0
                        return values;
        }

        public static double[] generateParetoNumbers(int n) {
                        /* Este método devuelve n-array de decimales con números aleatorios de pareto entre 0.0 y 1.0 */
                        double values[] = new double[n];
                        double[] U = new double[n];
                        double L, H, a;
```

```
                U = generateUniformNumbers(n); // matriz de numeros aleatorios de distribución unifor meentre
0 y 1
                L = 0.1;
                H = 1.0;
                a = 1.0;
                for (int i=0; i<U.length; i++)
                        values[i] = Math.pow( (-((U[i]*Math.pow(H,a) - U[i]*Math.pow(L,a) -
Math.pow(H,a))/(Math.pow(H,a)*Math.pow(L,a)))), (-1/a) );

                return values;

        }

        public static int[] randperm(int n) {
                /* Dado n, este devuelve una matriz de 1...n puestos en orden aleatorio */
                int[] values = new int[n];
                for (int i=0; i<n; i++)
                        values[i] = i+1;
                return nextPermutation(values);
        }

        public static int[] scalarMultiply(double scalar, double a[]) {
                /* Este método hace la multiplicación de un escalar por una matriz y devuelve una matriz
redondeada al entero más proximo superior */
                int[] b = new int[a.length];
                for (int i=0; i<a.length; i++)
                        b[i] = (int)(Math.ceil(a[i]*scalar));
                return b;
        }

        public static int[] one_comb(int n, int numberOfSelected) {
                /* Dado n, devuelve */
                int[] values = randperm(n);
                int[] selected = new int[numberOfSelected];
                for (int i=0; i<selected.length; i++)
                        selected[i] = values[i];
                return selected;
        }

        public static int[] nextPermutation(int a[]) {
                /* Este devuelve una permutación aleatoria de la matriz a*/
                int[] b = (int[])a.clone();
                for (int k = b.length - 1; k > 0; k--) {
                        int w = (int)Math.floor(Math.random() * (k+1));
                        int temp = b[w];
                        b[w] = b[k];
                        b[k] = temp;
                }
                return b;
        }

        public static int[] sort(int[] values) {
                boolean hayCambios = true;
                int tmp;
                while (hayCambios) {
                        hayCambios = false;
                        for (int j=0; j<values.length-1; j++) {
                                if (values[j] > values[j+1]) {
                                        tmp = values[j];
                                        values[j] = values[j+1];
                                        values[j+1] = tmp;
                                        hayCambios = true;
                                }
                        }
```

```
                }
                return values;
        }

}
```