TDTS04 Computer Networks and Distributed Systems

Distance Vector Routing Lab - Code

Authors
Maximilian Bragazzi Ihrén, maxbr431@student.liu.se
Markus Lewin, marle943@student.liu.se

VT1
Version 1.0

1. Audit Descriptions

Ver.	Audit Description	Date
1.0	Written for report.	150303

2. RouterNode.java

```
// Copyright maxbr431, marle943

import javax.swing.*;

public class RouterNode {
  private boolean POISONEDREVERSE = true;
  private int myID;
  private GuiTextArea myGUI;
  private RouterSimulator sim;
  //Which routers you are routing through
  private int[] throughWhich = new int[RouterSimulator.NUM_NODES];
  //Your direct costs
  private int[] directCosts = new int[RouterSimulator.NUM_NODES];
  //The entire table
  private int[][] allcosts =
    new int[RouterSimulator.NUM_NODES][RouterSimulator.NUM_NODES];

  //--------------------------------------------------
  public RouterNode(int ID, RouterSimulator sim, int[] costs) {
    myID = ID;
    this.sim = sim;
    myGUI = new GuiTextArea("  Output window for Router #"+ ID + "  ");

    //Default the table
    for (int x = 0; x < allcosts.length; ++x) {
      if (x == myID)
```

```
      continue; //skip self, will be set later
    for (int y = 0; y < allcosts[x].length; ++y) {
      if (x == y) {
        allcosts[x][y] = 0;
      } else {
        allcosts[x][y] = RouterSimulator.INFINITY;
      }
    }
  }

  allcosts[myID] = costs.clone();
  directCosts = costs.clone();

  for (int i = 0; i < throughWhich.length; ++i) {
    if (directCosts[i] == RouterSimulator.INFINITY) {
      //if not neighbor, default to inf
      throughWhich[i] = RouterSimulator.INFINITY;
    } else {
      throughWhich[i] = i;
    }
  }

  printDistanceTable();
  sendUpdateToAllNeighbors();
 }

 //--------------------------------------------------
 public void recvUpdate(RouterPacket pkt) {
  myGUI.println("Receiving packet from " + pkt.sourceid);

  boolean changes = false;
  for (int i = 0; i < pkt.mincost.length; ++i) {
    if (allcosts[pkt.sourceid][i] != pkt.mincost[i]) {
```

```
      allcosts[pkt.sourceid][i] = pkt.mincost[i];
      changes = true;
    }
  }

  if (changes) { //Recalculate all costs
    changes = false;

    for (int i = 0; i < RouterSimulator.NUM_NODES; ++i) {
      if (i == myID)
        continue;

      //If any cost changes occured
      if (allcosts[myID][i] !=
          (allcosts[throughWhich[i]][i] + allcosts[myID][throughWhich[i]])) {
        allcosts[myID][i] =
          allcosts[throughWhich[i]][i] + allcosts[myID][throughWhich[i]];
        changes = true;
      }

      //If directcost is cheapest
      if (directCosts[i] < allcosts[myID][i]) {
        allcosts[myID][i] = directCosts[i];
        throughWhich[i] = i;
        changes = true;
      }

      for (int j = 0; j < RouterSimulator.NUM_NODES; ++j) {
        if (allcosts[myID][j] > (allcosts[myID][i] + allcosts[i][j])) {
          allcosts[myID][j] = directCosts[i] + allcosts[i][j];
          throughWhich[j] = throughWhich[i];
          changes = true;
        }
```

```
      }
    }

    if (changes) {
      sendUpdateToAllNeighbors();
    }
  }

  printDistanceTable();
}


//---------------------------------------------------
private void sendUpdate(RouterPacket pkt) {
  myGUI.println("Sending packet to " + pkt.sourceid);
  sim.toLayer2(pkt);

}

private void sendUpdateToAllNeighbors() {
  for (int i = 0; i < directCosts.length; ++i) {
    if (directCosts[i] == RouterSimulator.INFINITY ||
        directCosts[i] == 0)
      continue; //Skip non-neighbors + self
    int[] send = allcosts[myID].clone();
    if (POISONEDREVERSE) {
      send = poisonedreverse(send, i);
    }

    sendUpdate(new RouterPacket(myID, i, send));
  }
}
```

```
 private int[] poisonedreverse(int[] send, int dest) {
   for (int i = 0; i < send.length; ++i) {
     if (throughWhich[i] == dest) {
       send[i] = RouterSimulator.INFINITY;
     }
   }
   return send;
 }



 //--------------------------------------------------
 public void printDistanceTable() {
   myGUI.println();
   myGUI.println("Current table for " + myID +
                 "  at time " + sim.getClocktime());
   myGUI.println();

   myGUI.println("=== TABLE ===");
   myGUI.print("Nodes\t|\t");
   for (int i = 0; i < RouterSimulator.NUM_NODES; ++i) {
     myGUI.print(i + "\t|\t");
   }
   myGUI.println();

   for (int i = 0; i < RouterSimulator.NUM_NODES + 1; ++i) {
     myGUI.print("================");
   }
   myGUI.println();

   for (int i = 0; i < RouterSimulator.NUM_NODES; ++i) {
     myGUI.print(i + "\t|\t");

     for (int j = 0; j < RouterSimulator.NUM_NODES; ++j) {
```

```
      myGUI.print(allcosts[i][j] + "\t|\t");
    }
    myGUI.println();
  }

  myGUI.println();
  myGUI.println("=== My Routes ===");
  myGUI.print("Nodes\t|\t");
  for (int i = 0; i < RouterSimulator.NUM_NODES; ++i) {
    myGUI.print(i + "\t|\t");
  }
  myGUI.println();

  for (int i = 0; i < RouterSimulator.NUM_NODES + 1; ++i) {
    myGUI.print("================");
  }
  myGUI.println();

  myGUI.print("Cost\t|\t");
  for (int i: allcosts[myID]) {
    myGUI.print(i + "\t|\t");
  }
  myGUI.println();

  myGUI.print("Through\t|\t");
  for (int i: throughWhich) {
    myGUI.print(i + "\t|\t");
  }
  myGUI.println();
 }

 //---------------------------------------------------
 public void updateLinkCost(int dest, int newcost) {
```

```
    myGUI.println("Link " + myID + " => " + dest + " updated to " + newcost + " from " +
directCosts[dest]);

    directCosts[dest] = newcost;
    //If we are routing directly to dest
    if (throughWhich[dest] == dest) {
      allcosts[myID][dest] = newcost;
    } else if (directCosts[dest] < allcosts[myID][dest]) {
      //If direct routing is cheaper than current
      allcosts[myID][dest] = directCosts[dest];
      throughWhich[dest] = dest;
    }

    for (int i = 0; i < RouterSimulator.NUM_NODES; ++i) {
      //If cheaper to route through change
      if (allcosts[myID][i] > allcosts[myID][dest] + allcosts[dest][i]) {
        allcosts[myID][i] = allcosts[myID][dest] + allcosts[dest][i];
        throughWhich[i] = throughWhich[dest];
      }
    }

    sendUpdateToAllNeighbors();
  }
}
```