1. Project Description

This project is about writing a network simulator application that simulates a routing algorithm, and may be extendable to simulate other network traffics or algorithms. The project initially started with a big feature list but reduced in scope to implementing the distance vector routing algorithm in a simple network simulator given at the minimum.

The simulator was taken from works of professors Anders Lindgren and Johan Nykvist (http://www.sm.luth.se/csee/courses/smd/123/current/labs/routing.html). The basic structure and classes of the simulator are as follows:

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DV</td>
<td>The main program.</td>
</tr>
<tr>
<td>Simulator</td>
<td>The simulator core. Handles scheduling and runs the different events.</td>
</tr>
<tr>
<td>Event</td>
<td>An abstract class that represents an event.</td>
</tr>
<tr>
<td>Link</td>
<td>A class that represents a unidirectional link between two routers.</td>
</tr>
<tr>
<td>Router</td>
<td>A class that represents a router.</td>
</tr>
<tr>
<td>RoutingEntry</td>
<td>A &quot;row&quot; in the routing table that contains information on how to reach a</td>
</tr>
<tr>
<td></td>
<td>certain destination (specifying the next hop router and what the cost to</td>
</tr>
<tr>
<td></td>
<td>reach the destination).</td>
</tr>
<tr>
<td>RoutingTable</td>
<td>Represents a routing table and contains a number of RoutingEntry-objects.</td>
</tr>
</tbody>
</table>

The tasks were to complete the implementation of the Router.updateRoutingTable() and RoutingTable.updateEntry() methods, so the routers in the simulated network would update their routing tables correctly. In addition, code may be added for loading external configuration files for different network configurations, provide graphical user interface frontend, extend the functionalities of the simulator, and so on.

The report must demonstrate the following from this project:

- What did you change?
- How did you implement the routing tables and algorithm?
- How did this implement an event driven simulation?
- Run with your own network configuration with at least 5 routers.
2. Design and Implementation

2.1. Changes Made

Besides the implementations of the two required methods, I attempted to expand the simple simulator with a more generic Node class which can be either a Router or a Host, handle other type of messages not only restricting to distance vector updates, and some more which I will describe briefly in upcoming sections. Hence, the overall structure of the project has some additional classes, although in the end many of them were mostly not much utilized.

Below are some of the changes made to the initial given simulator code:

- Class DV is renamed to NovaSim, to more accurately reflect the project name. NovaSim also extends JFrame to implement a GUI (described below). It provides a few static utility methods to write to the logging area of the GUI, popup a error message to user when something goes wrong, and save the logging area to a text file.

- Introduced an abstract Node class as a superclass to Router – this was done with the anticipation of implementing ‘hosts’; however, I was not able to get that far. Node contains basic information on the ‘name’ of the node and list of ‘interfaces’ (next bullet). An abstract receive(Message) and addNeighbor() methods were added to be implemented by the subclasses.

- Using the ‘name’ of a Node object, I replaced the integer values for router IDs with String name values. Some of the existing methods’ signature, e.g. RoutingTable.updateEntry(), have changed to use String instead of int.

- Added a new class Interface which stands between a Node and a Link, and provides the interface ID (e.g. IP address), in support for our initial in-class discussion. Adding these Interface objects weren’t trivial, because a lot of methods were tied between the direct relationship between a Router and Link object in this simple simulator. It ended up not being used, too, because I didn’t get to the point where I can extract information for individual routers from the GUI while the simulator was working.

- A Message class contains the source and destination IDs of the Interface (essentially the address) and an Object for the actual object. For the DV update message, this Object is an instance of a vector of RoutingEntry-s. The Router’s receive() method has some code for
handling the DV update message as well as other message (for the future).

- I took classmate Jon Bulava’s GUI implementation and reduced it to fit my purpose as the application frontend. In particular, the back button and the speed slider were taken out, and a menu-bar was added. The attempt was to make the simulator not run until a configuration is loaded and the start button is pressed; however, I did not complete the load configuration part, so I left it in a ‘debug’ mode where hard-coded code of the default configuration is run when the start button is pressed.

- A Network class is introduced that would contain all routers, links, and interfaces, and represent a single network configuration. The network object is correctly populated; only the configuration parsing part is not complete.

- NovaSim class has extra code to read from a XML configuration file and load the network. I intended to use a DOM parser from Apache Xerces, but trying it in a hurry didn’t implement it correctly and get a ClassCastException when trying to load the default configuration written in an XML file (See Appendix for the XML configuration).

- Also, attempted to add steps in the Simulator class and have a bit groundwork done. It’s not done and thus not linked to the GUI. There are a few items in the GUI not fully implemented though I tried to take out as many as possible (in time).

### 2.2. Routing Algorithm Implementation

The methods Router.updateRoutingTable() and RoutingTable.updateEntry() needed additional code in order to complete the routing simulator. The implementation of these methods were basically confined to how the original authors decided to organize them. Any changes to the structure of these two classes would have major effects, so I made my additions very minimal.

```java
public void Router.updateRoutingTable()
```

When the router receives an update message, the message is added to the member vector neighborDV, and this method is being called. The message is essentially a vector of RoutingEntry objects (sent by updateEntry() via Link objects as an Event). When this method is called neighborDV is scanned to check if any changes were made to the routing table. I
introduced a Boolean variable, `dvUpdated`, which is set to true once any update was made to the routing table during the scan, so at the end of the scan the router’s neighbors are notified of the update via `notifyNeighbors()`. Below is the code snippet of the `updateRoutingTable()` method:

```java
public void updateRoutingTable() {
    boolean dvUpdated = false;
    /*
    * The following loop will iterate through the set of DVs received from
    * all neighbors of this router.
    */
    Iterator iter = neighborsDVs.keySet().iterator();
    while (iter.hasNext()) {
        // THE FOLLOWING TWO VARIABLES SHOULD BE USEFUL TO YOU!
        String neighborID = (String) iter.next();
        Vector neighborDV = (Vector) neighborsDVs.get(neighborID);

        // Check each line in the received routing table
        for (int i = 0; i < neighborDV.size(); i++) {
            RoutingEntry entry = (RoutingEntry) neighborDV.get(i);
            String destination = entry.getDestination();
            int distance = entry.getDistance();

            boolean updated = routerDV.updateEntry(destination, distance, neighborID);
            if (updated)
                dvUpdated = true;
        }

        // Done with DV received from neighbor. Remove it from the map.
        iter.remove();
    }

    /*
    * If any change is made to the routing table, inform all neighbors
    */
    if (dvUpdated) {
        notifyNeighbors();
    }
}
```

```java
public void RouterTable.updateEntry()
```

This method is where the distance vector algorithm is implemented. To be more accurate, the underlying structures of these provided classes actually should changes. That is, a distance table would have to maintain all combinations of distance entries to a destination via a hop, and a forwarding table linking destination to the next hop only. What we have here is rather a mixture of these two tables, but it serves the purpose of this simple simulator. My plan was to enhance these classes and introduce a forwarding table as a mapping from destination to next hop; that way it would be accurately working when implementing dynamic link costs, etc.

Without changing the structure of the `RoutingEntry`-s, it was very simple to implement this method. The key is to compare the already existing distance (or cost) for a destination entry with
the newly calculated distance, which is the distance to the source router of the message and its
distance vector value. If the newly computed value is smaller, we replace the RoutingEntry
object’s distance and nexthop values with the new values. Below is the code:

```java
public boolean updateEntry(String destination, int distance, String nexthop) {
    // find destination from table
    RoutingEntry entry = getEntry(destination);
    RoutingEntry neighbor = getEntry(nexthop);
    ...  
    int newdistance = distance + getEntry(nexthop).getDistance();
    // if not found, add entry and return true
    if (entry == null) {
        addEntry(destination, newdistance, nexthop);
        return true;
    }
    // if found, check if 'distance' is less than the current entry.
    // If yes, update table entry, and return true
    if (newdistance < entry.getDistance()) {
        entry.setDistance(newdistance);
        entry.setNexthop(nexthop);
        return true;
    }
    return false;
}
```

2.3. Event-Driven Simulation

The mechanism used in this simulator for event-driven simulation is rather simple.

- The Simulator object is declared public static in the main class. Once the JVM runs the
  program all other objects in the same JVM has access to it statically.

- The Simulator maintains a Vector of Event objects. When the simulator starts, it pops out
  one Event object at a time, updates current clock to the Event time and call its handle() method. The simulator runs until no events are left in the event queue (vector).

- There is one concrete Event type in the basic simulator, DeliverEvent, that is ‘scheduled’
  with the simulator when a message is sent to a Link object. This is done when a router
determines to send notifications of DV changes to its neighbors. The time for the delivery
event is chosen to add some random delay (between 0.0 and 1.0) in addition to the default
delay of 0.3. The simulator is using double as unit for the time, probably because it’s easy
to use with Math.random().

- For an example, the DeliverEvent.handle() method calls Link.deliver() method to pass the
  message to the destination router’s receive() method, so it can decide what to do.
2.4. Running my own configuration

Unfortunately, I did not get to the point in testing out my own configuration, but I made some additions to the default configuration (with Links having IP addresses) and were able to see the same results. An XML file for the default configuration was also made (see Appendix).

3. Execution of Program and Test Results

1. Initial GUI launch screen:
2. Running the simulation of the default configuration, using the Start (>) button:

3. Saving the outputted log area:
4. Clearing the log area via the menu item:

5. Loading configuration XML file fails:
6. Occasionally, running the default configuration showed the below error message (coded by myself). It appeared at the code where the neighbors’ distance function was retrieved from the distance vector table. I have not been able to debug this code yet, but it probably has to do with the randomized events occurrences. I am removing distance vectors stored in the variable neighborDV, so the router only checks for update if necessary. Due to the randomized timing of the events, it may be that a RouterEntry is removed from this vector while trying to get the distance to this vector.
4. Conclusion

This project was by far the largest in scope for the semester; I regret that I didn’t have enough time to work on it although this was available to us for quite a long time. The initial attempt to work in different teams on different modules and collaborating the work seemed like a good idea; but realistically, only a few could afford to do that and only few actually were actively participating. However, I also had good help from such fellow classmates, and they in no doubt deserve extra credit for their work.

Despite the original purpose of this project, I believe I learned little about routing algorithm from this project, mostly due to the many side-work associated to get this project to something useful. However, it was a good project to learn and get hands on event-driven network simulator; software design and other Java technology to get the work done. Since the baseline simulator program were tightly coupled within its own classes and was not meant for expandability, enhancing the simulator would have actually required major redesign and implementation.
5. **Bibliography**

Credit goes to professors Anders Lindgren and Johan Nykvist for their work on the basic simulator (http://www.sm.luth.se/csee/courses/smd/123/current/labs/routing.html). The GUI layout and icons were taken from Jon Bulava’s posted source code.
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileWriter;
import java.io.PrintWriter;
import java.util.Hashtable;
import java.util.Vector;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.border.Border;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;
import javax.swing.filechooser.FileFilter;
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

/**
 * This class initializes and executes a simulation.
 * @author Anders Lindgren
 * @author Johan Nykvist
 */
public class NovaSim extends JFrame implements ActionListener {

    public final static boolean debug_flag = true;

    private static NovaSim instance = null;

    public static Simulator sim = new Simulator();

    public static Network net = null;

    public final static String FILE_TYPE = "*.xml";

    private static final String DEBUG_FILENAME = "debug.txt";

    /**
     * NovaSim instance for making the class singleton
     * @return
     */
    private static NovaSim getInstance() {
        if (null == instance) {
            instance = new NovaSim();
        }
        return instance;
    }

    /**
     * This method initializes a new simulation.
     * After this method returns, the simulation is ready to start.
     * @param sim
     */
    public static void initSimulation(Simulator sim) {
        NovaSim.getInstance().initSimulation(sim);
    }

    /**
     * Initialization method.
     * Additional methods can be added here.
     */
    public void initSimulation(Simulator sim) {
        this.sim = sim;

        NodeList networks = sim.getNetworks();
        Node network = networks.item(0);

        // Create new network
        Network newNetwork = new Network();
        sim.addNetwork(newNetwork);

        // Add nodes and links to the new network
        sim.addNodes(network, newNetwork);
        sim.addLinks(network, newNetwork);

        // Schedule simulation events
        sim.scheduleEvents(newNetwork);

        // Start simulation
        sim.start();

        // Save configuration
        File file = new File(DEBUG_FILENAME);
        PrintWriter writer = new PrintWriter(file);
        writer.println(sim.saveConfiguration());
        writer.close();
    }

    public static void main(String[] args) {
        // Create new simulation
        NovaSim.getInstance().initSimulation(new Simulator());
    }
}
private JFileChooser fileChooser;
public boolean initialized = false;
public boolean running = false;

/* GUI components */
JMenuBar menuBar;
JPanel controlPanel;
JPanel rightPanel;
JPanel displayPanel;
JPanel logPanel;
JTextArea logArea;

/** Private constructor to make NovaSim singleton */
private NovaSim() {
    super("NovaSim - Charlie Han's routing simulation at Villanova");
}

/** returns the NovaSim singleton instance */
public static NovaSim getInstance() {
    if (instance == null)
        instance = new NovaSim();
    return instance;
}

/* Public static methods */
public static void log(Object str) {
    if (instance == null || !instance.initialized)
        return;
    instance.logArea.append(str + "\n");
}

public static void clearLog() {
    if (instance == null || !instance.initialized)
        return;
    instance.logArea.setText("");
}

public static void error(String str) {
    log("[ERROR] " + str);
}

public static void showError(String str) {
    if (instance == null || !instance.initialized)
        return;
    error(str);
    JOptionPane.showMessageDialog(instance, str, "ERROR", JOptionPane.ERROR_MESSAGE);
}

public static void debug(String str) {
    if (debug_flag)
        log("[DEBUG] " + str);
}

/**
 * Creates GUI - took code written by Jon Bulava
 * and modified to meet my need
 */
public void createAndShowGUI() {
    //Create and set up the window.
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setPreferredSize(new Dimension(800, 600));
    // Menus
    menuBar = new JMenuBar();
    setJMenuBar(menuBar);
    JMenu fileMenu = new JMenu("File");
JMenuItem loadMenuItem = new JMenuItem("Load Configuration");
JMenuItem saveMenuItem = new JMenuItem("Save Log");
JMenuItem clearMenuItem = new JMenuItem("Clear Log");
JMenuItem exitMenuItem = new JMenuItem("Exit");
loadMenuItem.setActionCommand("Load Configuration");
saveMenuItem.setActionCommand("Save Log");
clearMenuItem.setActionCommand("Clear Log");
exitMenuItem.setActionCommand("Exit");
loadMenuItem.addActionListener(this);
saveMenuItem.addActionListener(this);
clearMenuItem.addActionListener(this);
exitMenuItem.addActionListener(this);
fileMenu.add(loadMenuItem);
fileMenu.addSeparator();
fileMenu.add(saveMenuItem);
fileMenu.add(clearMenuItem);
fileMenu.addSeparator();
fileMenu.add(exitMenuItem);
menuBar.add(fileMenu);

JMenu runMenu = new JMenu("Run");
JMenuItem startMenuItem = new JMenuItem("Start");
JMenuItem stepMenuItem = new JMenuItem("Step");
JMenuItem stopMenuItem = new JMenuItem("Stop");
startMenuItem.setActionCommand("Start");
stepMenuItem.setActionCommand("Step");
stopMenuItem.setActionCommand("Stop");
startMenuItem.addActionListener(this);
stepMenuItem.addActionListener(this);
stopMenuItem.addActionListener(this);
runMenu.add(startMenuItem);
runMenu.add(stepMenuItem);
runMenu.add(stopMenuItem);
menuBar.add(runMenu);

JMenu helpMenu = new JMenu("Help");
JMenuItem aboutMenuItem = new JMenuItem("About NovaSim");
helpMenu.add(aboutMenuItem);
menuBar.add(helpMenu);

// Main Panels
controlPanel = new JPanel(new GridLayout(0,1));
rightPanel   = new JPanel(new GridLayout(0,1));
displayPanel = new JPanel(new GridLayout(0,1));
logPanel     = new JPanel(new GridLayout(0,1));

// Common Border
//Border blackline = BorderFactory.createLineBorder(Color.black);
Border blackline = BorderFactory.createEtchedBorder(EtchedBorder.LOWERED);

// --- Set up display panel ---
JLabel displayLbl = new JLabel(" ");
displayPanel.add(displayLbl);

// --- Set up log panel ---
JTextArea logArea = new JTextArea("Please load a configuration file... ");
JScrollPane scrollPane = new JScrollPane(logArea, JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
scrollPane.setPreferredSize(new Dimension(600, 200));
logPanel.add(scrollPane, BorderLayout.CENTER);

// --- Set up control panel ---
controlPanel.setLayout(new BoxLayout(controlPanel, BoxLayout.PAGE_AXIS));
controlPanel.setPreferredSize(new Dimension(200, 800));

// --- Set up file chooser ---
fileChooser = new JFileChooser();

// Control Panel - Config section
JPanel configPanel = new JPanel(new GridLayout(0,1));
TitledBorder titled = BorderFactory.createTitledBorder(blackline, "Configuration");
configPanel.setBorder(titled);
configPanel.setLayout(new FlowLayout());
JButton loadNew = new JButton("Load New... ");
loadNew.setActionCommand("Load Configuration ");
loadNew.addActionListener(this);
configPanel.add(loadNew);

// Control Panel - Options section
JPanel optionsPanel = new JPanel(new GridLayout(0,1));
titled = BorderFactory.createTitledBorder(blackline, "Options ");
optionsPanel.setBorder(titled);
optionsPanel.setLayout(new FlowLayout(FlowLayout.LEFT, 10, 5 ));

JRadioButton fullSimButton = new JRadioButton("Full Simulation ");
JRadioButton stepSimButton = new JRadioButton("Run for ");
JTextField stepNumber = new JTextField(3 );
JLabel stepsLine = new JLabel("steps. ");
fullSimButton.setSelected(true);
ButtonGroup group = new ButtonGroup();
group.add(fullSimButton);
group.add(stepSimButton);

optionsPanel.add(fullSimButton);
optionsPanel.add(stepSimButton);
optionsPanel.add(stepNumber);
optionsPanel.add(stepsLine);

// Control Panel - Run Controls section
JPanel runPanel = new JPanel(new GridLayout(0,1));
titled = BorderFactory.createTitledBorder(blackline, "Controls ");
r
runPanel.setBorder(titled);
runPanel.setLayout(new FlowLayout());

ImageIcon stopicon = new ImageIcon("images/stop.gif ");
ImageIcon forwardicon = new ImageIcon("images/stepforward.gif ");
ImageIcon runicon = new ImageIcon("images/run.gif ");

JButton stop = new JButton(stopicon);
JButton forward = new JButton(forwardicon);
JButton run = new JButton(runicon);

stop.setToolTipText("Stop");
stop.setActionCommand("Stop");
stop.addActionListener(this);
forward.setToolTipText("Step Forward");
forward.setActionCommand("Step");
forward.addActionListener(this);
run.setToolTipText("Run");
run.setActionCommand("Start");
run.addActionListener(this);

runPanel.add(stop);
runPanel.add(run);
runPanel.add(forward);

// Control Panel - Log options section
JPanel logOptPanel = new JPanel(new GridLayout(0,1));
titled = BorderFactory.createTitledBorder(blackline, "Log Options ");
logOptPanel.setBorder(titled);
logOptPanel.setLayout(new BoxLayout(logOptPanel, BoxLayout.X_AXIS));

JButton clearLog = new JButton("Clear Log");
JButton saveLog = new JButton("Save Log");
clearLog.addActionListener(this);
clearLog.setActionCommand("Clear Log");
saveLog.addActionListener(this);
saveLog.setActionCommand("Save Log");

logOptPanel.add(clearLog);
logOptPanel.add(Box.createHorizontalGlue());
logOptPanel.add(saveLog);
// Add section panels to Control Panel  
controlPanel.add(configPanel);  
controlPanel.add(optionsPanel);  
controlPanel.add(runPanel);  
controlPanel.add(logOptPanel);  
// Create Split Pane for Right Panel  
JSplitPane splitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT, displayPanel, logPanel);  
splitPane.setOneTouchExpandable(true);  
splitPane.setResizeWeight(1);  
splitPane.setDividerLocation(0);  
rightPanel.add(splitPane);  
// --- Add panels to main frame ---  
getContentPane().add(controlPanel, BorderLayout.WEST);  
getContentPane().add(rightPanel, BorderLayout.CENTER);  

// Display the window.  
pack();  
setVisible(true);  
initialized = true;  
}  

public void actionPerformed(ActionEvent event) {  
String action = event.getActionCommand();  
if ("Start".equals(action)) {  
start();  
}  
else if ("Stop".equals(action)) {  
stop();  
}  
else if ("Step".equals(action)) {  
}  
else if ("Clear Log".equals(action)) {  
clearLog();  
}  
else if ("Save Log".equals(action)) {  
saveFile(logArea.getText(), "txt");  
}  
else if ("Load Configuration".equals(action)) {  
load("xml");  
}  
else if ("Exit".equals(action)) {  
}  
}  

public void start() {  
if (running) {  
showError("Stop the current simulation before starting a new one");  
return;  
}  
if (net == null) && !debug_flag) {  
showError("Please load a configuration in order to start simulation");  
return;  
}  
running = true;  
clearLog();  
if (debug_flag) {  
net = new Network();  
// * Create router instances and define their interfaces */  
Router rl = new Router("17");  
rl.addInterface("192.168.0.101", "255.255.255.0");  
rl.addInterface("10.0.1.11", "255.255.255.128");  
}  
public void stop() {  
if (running) {  
showError("Stop the current simulation before stopping a new one");  
return;  
}  
if (net == null) && !debug_flag) {  
showError("Please load a configuration in order to stop simulation");  
return;  
}  
running = false;  
clearLog();  
if (debug_flag) {  
net = new Network();  
// * Create router instances and define their interfaces */  
Router rl = new Router("17");  
rl.addInterface("192.168.0.101", "255.255.255.0");  
rl.addInterface("10.0.1.11", "255.255.255.128");  
}  
public void stop() {  
if (running) {  
showError("Stop the current simulation before stopping a new one");  
return;  
}  
if (net == null) && !debug_flag) {  
showError("Please load a configuration in order to stop simulation");  
return;  
}  
running = false;  
clearLog();  
if (debug_flag) {  
net = new Network();  
// * Create router instances and define their interfaces */  
Router rl = new Router("17");  
rl.addInterface("192.168.0.101", "255.255.255.0");  
rl.addInterface("10.0.1.11", "255.255.255.128");  
}  
public void stop() {  
if (running) {  
showError("Stop the current simulation before stopping a new one");  
return;  
}  
if (net == null) && !debug_flag) {  
showError("Please load a configuration in order to stop simulation");  
return;  
}  
running = false;  
clearLog();  
if (debug_flag) {  
net = new Network();  
// * Create router instances and define their interfaces */  
Router rl = new Router("17");  
rl.addInterface("192.168.0.101", "255.255.255.0");  
rl.addInterface("10.0.1.11", "255.255.255.128");  
}
r1.addInterface("222.111.0.1", "255.255.255.192");
r1.addInterface("111.0.222.1", "255.255.254.0");
r1.addInterface("100.200.10.21", "255.255.252.0");
net.addRouter(r1);

Router r2 = new Router("2");
r2.addInterface("192.168.0.102", "255.255.255.0");
r2.addInterface("10.0.1.12", "255.255.255.128");
r2.addInterface("222.111.0.2", "255.255.255.192");
r2.addInterface("111.0.222.2", "255.255.254.0");
r2.addInterface("100.200.10.22", "255.255.252.0");
net.addRouter(r2);

Router r3 = new Router("3");
r3.addInterface("192.168.0.103", "255.255.255.0");
r3.addInterface("10.0.1.13", "255.255.255.128");
r3.addInterface("222.111.0.3", "255.255.255.192");
r3.addInterface("111.0.222.3", "255.255.254.0");
r3.addInterface("100.200.10.23", "255.255.252.0");
net.addRouter(r3);

Router r4 = new Router("4");
r4.addInterface("192.168.0.104", "255.255.255.0");
r4.addInterface("10.0.1.14", "255.255.255.128");
r4.addInterface("222.111.0.4", "255.255.255.192");
r4.addInterface("111.0.222.4", "255.255.254.0");
r4.addInterface("100.200.10.24", "255.255.252.0");
net.addRouter(r4);

Router r5 = new Router("5");
r5.addInterface("192.168.0.105", "255.255.255.0");
r5.addInterface("10.0.1.15", "255.255.255.128");
r5.addInterface("222.111.0.5", "255.255.255.192");
r5.addInterface("111.0.222.5", "255.255.254.0");
r5.addInterface("100.200.10.25", "255.255.252.0");
net.addRouter(r5);

Router r6 = new Router("6");
r6.addInterface("192.168.0.106", "255.255.255.0");
r6.addInterface("10.0.1.16", "255.255.255.128");
r6.addInterface("222.111.0.6", "255.255.255.192");
r6.addInterface("111.0.222.6", "255.255.254.0");
r6.addInterface("100.200.10.26", "255.255.252.0");
net.addRouter(r6);

/*
 * Define the topology by connecting the nodes' interfaces
 */
net.connect(r1.getInterface("192.168.0.101"), r5.getInterface("192.168.0.105"), 10);
net.connect(r5.getInterface("192.168.0.105"), r1.getInterface("192.168.0.101"), 10);
net.connect(r1.getInterface("222.111.0.1"), r3.getInterface("222.111.0.3"), 6);
net.connect(r3.getInterface("222.111.0.3"), r1.getInterface("222.111.0.1"), 6);
net.connect(r2.getInterface("222.111.0.2"), r6.getInterface("222.111.0.6"), 13);
net.connect(r6.getInterface("222.111.0.6"), r2.getInterface("222.111.0.2"), 13);
net.connect(r2.getInterface("111.0.222.2"), r5.getInterface("111.0.222.5"), 11);
net.connect(r5.getInterface("111.0.222.5"), r2.getInterface("111.0.222.2"), 11);
net.connect(r2.getInterface("100.200.10.22"), r3.getInterface("100.200.10.23"), 14);
net.connect(r3.getInterface("100.200.10.23"), r2.getInterface("100.200.10.22"), 14);
net.connect(r3.getInterface("100.200.10.23"), r4.getInterface("100.200.10.24"), 5);
net.connect(r4.getInterface("100.200.10.24"), r3.getInterface("100.200.10.23"), 5);
net.connect(r3.getInterface("10.0.1.13"), r5.getInterface("10.0.1.15"), 17);
net.connect(r5.getInterface("10.0.1.15"), r3.getInterface("10.0.1.13"), 17);
net.connect(r4.getInterface("222.111.0.4"), r5.getInterface("222.111.0.5"), 7);
net.connect(r5.getInterface("222.111.0.5"), r4.getInterface("222.111.0.4"), 7);
net.connect(r4.getInterface("192.168.0.104"), r6.getInterface("192.168.0.106"), 16);
net.connect(r6.getInterface("192.168.0.106"), r4.getInterface("192.168.0.104"), 16);
net.connect(r5.getInterface("100.200.10.25"), r6.getInterface("100.200.10.26"), 8);
net.connect(r6.getInterface("100.200.10.26"), r5.getInterface("100.200.10.25"), 8);
}

 /**<
 * Run the simulation
 */
log("Simulation starts...");

 /**<
 * Each router notifies its neighbors to get the ball rolling
 */
for (Router router : net.getRouters().values()) {
    router.notifyNeighbors();
}
sim.start();
log("Simulation ended!");

 /**<
 * simulation has ended, print the final routing solution
 */
log("The stable routing solution is:");
// Print each router's routing table
for (Router router : net.getRouters().values()) {
    log(router);
}
running = false;

public void stop() {
    if (!running)
        return;

    log("Stopping simulation as per user's request");
sim.stop();
    running = false;
}

/** gets the extension of the filename */
private String getExtension(String filename) {
    String ext = null;
    int i = filename.lastIndexOf('.');
    if (i > 0 && i < filename.length() - 1) {
        ext = filename.substring(i+1).toLowerCase();
    }
    return ext;
}

/** saves a text file with the specified file extension */
private void saveFile(String text, final String filterExtension) {
    fileChooser.setFileFilter(new FileFilter() {
        public boolean accept(File f) {
            if (f.isDirectory()) {
                return true;
            }
            return filterExtension.equalsIgnoreCase(getExtension(f.getName()));
        }
        public String getDescription() { return filterExtension; }
    });
    fileChooser.setName("Save Log File");
    int returnVal = fileChooser.showOpenDialog(this);
    if (returnVal == JFileChooser.APPROVE_OPTION) {
        File file = fileChooser.getSelectedFile();
        fileChooser.setFileFilter(new FileFilter() {
            public boolean accept(File f) {
                if (f.isDirectory()) {
                    return true;
                }
                return filterExtension.equalsIgnoreCase(getExtension(f.getName()));
            }
            public String getDescription() { return filterExtension; }
        });
        fileChooser.setName("Save Log File");
        int returnVal = fileChooser.showOpenDialog(this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            File file = fileChooser.getSelectedFile();
            // Save the text to the file
        }
    }
}
if (!filterExtension.equalsIgnoreCase(getExtension(file.getName())))
    file = new File(file.getAbsolutePath() + "." + filterExtension.toLowerCase());

debug("Opening file " + file.getAbsolutePath() + " for writing the logs");
try {
    FileWriter fw = new FileWriter(file);
    PrintWriter pw = new PrintWriter(fw);
    pw.println(text);
    pw.close();
} catch (Exception e) {
    showError("Failed to write log to file: " + e.getMessage());
    return;
}

public void load(final String filterExtension) {
    fileChooser.setFileFilter(new FileFilter() {
        public boolean accept(File f) {
            if (f.isDirectory()) {
                return true;
            }
            return filterExtension.equalsIgnoreCase(getExtension(f.getName()));
        }
        public String getDescription() { return filterExtension; }
    }) ;
    fileChooser.setName("Load Configuration");
    int returnVal = fileChooser.showOpenDialog(this);
    if (returnVal != JFileChooser.APPROVE_OPTION)
        return;

    File file = fileChooser.getSelectedFile();
    String url = "file:" + file.getAbsolutePath();
    NovaSim.log("Loading configuration \[" + file.getAbsolutePath() + "] ... ");

    Network network = new Network();
    try {
        DOMParser parser = new DOMParser();
        parser.parse(url);
        Document doc = parser.getDocument();
        NodeList nodes = doc.getElementsByTagName("node");
        for (int i = 0; i < nodes.getLength(); i++) {
            Element node = (Element) nodes.item(i);
            boolean validNode = buildNode(network, node);
            if (!validNode) {
                NovaSim.showError("Invalid configuration. See log screen for details");
                return;
            }
        }

        NodeList links = doc.getElementsByTagName("link");
        for (int i = 0; i < links.getLength(); i++) {
            Element link = (Element) links.item(i);
            boolean validLink = buildLink(network, link);
            if (!validLink) {
                NovaSim.showError("Invalid configuration. See log screen for details");
                return;
            }
        }
    } catch (Exception ex) {
        NovaSim.error(ex.toString());
        NovaSim.showError("Error occurred while loading configuration. See logs for details.");
    }
}
private boolean buildNode(Network network, Element nodeElement) {
    boolean isRouter = false;
    String name = null;
    Vector intf = new Vector();

    Element child = (Element) nodeElement.getFirstChild();
    while (child != null) {
        String nodeName = child.getNodeName();
        if ("name".equals(nodeName)) {
            name = child.getNodeValue();
        } else if ("type".equals(nodeName)) {
            isRouter = "router".equals(child.getNodeValue());
        } else if ("interface".equals(nodeName)) {
            Vector ifInfo = buildInterface(child, name);
            if (ifInfo == null)
                return false;
            intf.add(ifInfo);
        } else if ("display".equals(nodeName) || "error".equals(nodeName)) {
            // TODO: implement more features
        } else {
            NovaSim.error("Unknown element \"+nodeName
            +\" found in node configuration");
            return false;
        }
        child = (Element) child.getNextSibling();
    }
    if (!isRouter)
        return true; // TODO: support host nodes too
    if (name == null || intf.isEmpty()) {
        NovaSim.error("Insufficient data for node element");
        return false;
    }
    // Build the actual Router object
    Router router = new Router(name);
    for (Object intf : intf)
        router.addInterface((String) intf.get(0), (String) intf.get(1));
    return network.addRouter(router);
}

private Vector buildInterface(Element ifElement, String node) {
    String id = null;
    String mask = null;
    Element child = (Element) ifElement.getFirstChild();
    while (child != null) {
        String nodeName = child.getNodeName();
        if ("id".equals(nodeName)) {
            id = child.getNodeValue();
        } else if ("mask".equals(nodeName)) {
            mask = child.getNodeValue();
        } else if ("error".equals(nodeName)) {
            // TODO: implement more features
        } else {
            NovaSim.error("Unknown element \"+nodeName
            +\" found in interface configuration");
            return null;
        }
        child = (Element) child.getNextSibling();
    }
    if (id == null || mask == null)
        return null;
Vector v = new Vector(2);
    v.add(id);
    v.add(mask);

    return v;
}

private boolean buildLink(Network network, Element linkElement) {
    int cost = -1;
    String from = null;
    String to = null;

    Element child = (Element) linkElement.getFirstChild();
    while (child != null) {
        String nodeName = child.getNodeName();
        if ("cost".equals(nodeName)) {
            cost = new Integer(child.getNodeValue());
        } else if ("from".equals(nodeName)) {
            from = child.getNodeValue();
        } else if ("to".equals(nodeName)) {
            to = child.getNodeValue();
        } else if ("display".equals(nodeName) || "error".equals(nodeName)) {
            // TODO: implement more features
        } else {
            NovaSim.error("Unknown element " + nodeName
                        + " found in node configuration");
            return false;
        }
        child = (Element) child.getNextSibling();
    }

    if (cost < 1 || from == null || to == null) {
        NovaSim.error("Insufficient data for link element");
        return false;
    }

    // Build the actual Link object
    Interface iffrom = network.getInterface(from);
    Interface ifto = network.getInterface(to);
    return network.connect(iffrom, ifto, cost);
}

/**
 * Initializes and launches simulation GUI
 * @param args run-time arguments
 */
public static void main(String[] args) {
    getInstance().createAndShowGUI();
}
public abstract class Event {

    private double time;

    public abstract void handle(double time);

    public double getTime() {
        return time;
    }

    public void setTime(double time) {
        this.time = time;
    }
}
public class Interface {
    private String id;
    private String mask;
    private Node node;

    public Interface(String id, String mask, Node node) {
        this.id = id;
        this.mask = mask;
        this.node = node;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getMask() {
        return mask;
    }

    public void setMask(String mask) {
        this.mask = mask;
    }

    public Node getNode() {
        return node;
    }

    public void setNode(Node node) {
        this.node = node;
    }
}
import java.util.LinkedList;
import java.util.List;
import java.util.Vector;

/**
 * This class models an unidirectional link with randomized delay.
 *
 * @author Anders Lindgren
 * @author Johan Nykvist
 */
public class Link {

    private Node sourceNode;
    private Node destinationNode;
    private int    cost;
    private double delay;
    private List   messages;

    /**
     * Creates a link instance.
     *
     * @param sourceNode
     *            the source node (sender)
     * @param destinationNode
     *            the destination node (receiver)
     * @param cost
     *            link cost
     */
    public Link(Node sourceNode, Node destinationNode, int cost) {
        this.destinationNode = destinationNode;
        this.sourceNode = sourceNode;
        this.cost = cost;
        this.delay = 0.3;
        this.messages = new LinkedList();
    }

    /**
     * Schedules and delivers the specified message to the destination node.
     *
     * @param message
     *            the message
     */
    public void send(Message message) {
        messages.add(message);
        NovaSim.sim.schedule(new DeliverEvent(this), NovaSim.sim.clock() + delay + Math.random());
    }

    /**
     * @return the link cost
     */
    public int getCost() {
        return cost;
    }

    /**
     * @return the destination node
     */
    public Node getDestinationNode() {
        return destinationNode;
    }

    private void deliver() {
        //Vector message = (Vector) messages.remove(0);
        //destinationNode.receive(message, sourceNode.getRouterID());
    }
}
private class DeliverEvent extends Event {
    Link link;
    DeliverEvent(Link link) {
        this.link = link;
    }

    /**
     * @param time *
     */
    public void handle(double time) {
        link.deliver();
    }
}
public class Message {
    protected Object msgObj;
    protected String srcId;
    protected String destId;
    // TODO: add lifetime
    
    public Message(Object msg, String src, String dest) {
        this.msgObj = msg;
        this.srcId = src;
        this.destId = dest;
    }
    
    public Object getMessage() {
        return msgObj;
    }
    
    public String getSource() {
        return srcId;
    }
    
    public String getDestination() {
        return destId;
    }
}
Appendix F – Network.java

```java
import java.util.TreeMap;

public class Network {

    /** maps router name to router object */
    private TreeMap<String, Router> routers = new TreeMap<String, Router>();

    /** maps source id of node interface (IP address) to its connected link */
    private TreeMap<String, Link> links = new TreeMap<String, Link>();

    /** maps id of node interface (IP address) to the link object */
    private TreeMap<String, Interface> interfaces = new TreeMap<String, Interface>();

    /** Identifying name of the network */
    //private String name;

    public TreeMap<String, Interface> getInterfaces() {
        return interfaces;
    }

    public Interface getInterface(String id) {
        return interfaces.get(id);
    }

    public TreeMap<String, Link> getLinks() {
        return links;
    }

    public TreeMap<String, Router> getRouters() {
        return routers;
    }

    public boolean addRouter(Router rt) {
        String name = rt.getName();
        if (routers.containsKey(name)) {
            NovaSim.error("Duplicate router entry found. Router \" + name + \" is already configured!");
            return false;
        }
        NovaSim.debug("Adding router \" + name + \" and its interfaces");
        for (Interface ifobj : rt.getInterfaces().values()) {
            interfaces.put(ifobj.getId(), ifobj);
        }
        routers.put(name, rt);
        return true;
    }

    public boolean connect(Interface iffrom, Interface ifto, int cost) {
        /* In reality, there should be a check here whether the two ends are in the same network */
        if (iffrom == null || ifto == null) {
            NovaSim.error("Link specification has wrong interface pointers, with uplink cost \" + cost + \"");
            return false;
        }

        // Check for duplicate links
        String fromId = iffrom.getId();
        if (links.containsKey(fromId)) {
            NovaSim.error("Link starting from interface \" + fromId + \" already exist!");
            return false;
        }
    }
}
```
/ Create the link
Node n1 = iffrom.getNode();
Node n2 = ito.getNode();
Link link = n1.addNeighbor(n2, cost);

links.put(iffrom.getId(), link);
return true;
}
import java.util.Hashtable;
import java.util.Vector;

/**
 * Abstract Node class
 * <p>
 * Defines the basic attributes and methods for a node type.
 * *
 * @author Charlie Han
 */
public abstract class Node {
    protected String name;
    protected Hashtable<String, Interface> interfaces;

    /** Constructor */
    public Node(String name) {
        this.name = name;
        this.interfaces = new Hashtable<String, Interface>();
    }

    /** Getters and Setters */
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Hashtable<String, Interface> getInterfaces() {
        return interfaces;
    }

    public void setInterfaces(Hashtable<String, Interface> interfaces) {
        this.interfaces = interfaces;
    }

    /** Adds an 'interface' between a node and a link */
    public Interface addInterface(String id, String mask) {
        if (interfaces.containsKey(id))
            return null;
        return interfaces.put(id, new Interface(id, mask, this));
    }

    public Interface getInterface(String id) {
        return interfaces.get(id);
    }

    /* Abstract methods to be implemented by subclasses */
    public abstract void receive(Message msg);
    public abstract Link addNeighbor(Node neighbor, int cost);
}
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
import java.util.Map;
import java.util.Vector;

/**
 * This class models a router that executes a DV-routing protocol.
 *
 * @author Anders Lindgren
 * @author Johan Nykvist
 */
public class Router extends Node {

    /**
     * The router's ID number.
     */
    private int routerID;

    /**
     * All links that connect the router to its neighbors.
     */
    private Vector links;

    /**
     * The router's DV routing table.
     */
    private RoutingTable routerDV;

    /**
     * Routing tables received from the router's neighbors.
     */
    private Map neighborsDVs;

    /**
     * Constructs a new router instance with the specified ID number.
     *
     * @param id the router's ID number
     */
    public Router(String id) {
        super(id+"");  /* temp */
        routerDV = new RoutingTable();
        links = new Vector();
        neighborsDVs = new HashMap();
        routerDV.addEntry(id, 0, id); // add an entry to the router itself
    }

    /**
     * Connects this router to the specified router through a link.
     *
     * @param neighbor the neighboring router
     * @param cost the link cost
     */
    public Link addNeighbor(Node neighbor, int cost) {
        Link link = new Link(this, neighbor, cost);
        links.addElement(link);
        routerDV.addEntry(neighbor.getName(), cost, neighbor.getName());
        return link;
    }

    /**
     * Stores the specified DV in the set of neighbors' DVs. This method is
     * called when a DV is received from a neighbor.
     *
     * @param receivedDV the neighbor's routing table
     * @param neighborID the neighbor's ID number
     */
    private void receive(Vector receivedDV, String neighborID) {

    }
}
neighborsDVs.put(neighborID, receivedDV);
updateRoutingTable();
}

/**
* Updates the routing table such that the path with the shortest distance
* is used for each destination. The router's neighbors should be notified
* if, and only if, changes are made to the routing table.
*/
public void updateRoutingTable()
{
    boolean dvUpdated = false;

    /**
     * The following loop will iterate through the set of DVs received from
     * all neighbors of this router.
     */
    Iterator iter = neighborsDVs.keySet().iterator();
    while (iter.hasNext()) {
        // THE FOLLOWING TWO VARIABLES SHOULD BE USEFUL TO YOU!
        String neighborID = (String) iter.next();
        Vector neighborDV = (Vector) neighborsDVs.get(neighborID);

        // Check each line in the received routing table
        for (int i = 0; i < neighborDV.size(); i++) {
            RoutingEntry entry = (RoutingEntry) neighborDV.get(i);
            String destination = entry.getDestination();
            int distance = entry.getDistance();
            boolean updated = routerDV.updateEntry(destination, distance, neighborID);
            if (updated)
                dvUpdated = true;
        }

        // Done with DV received from neighbor. Remove it from the map.
        iter.remove();
    }

    // If any change is made to the routing table, inform all neighbors
    if (dvUpdated) {
        notifyNeighbors();
    }
}

/**
* Sends the router's DV to all neighbors.
*/
public void notifyNeighbors()
{
    NovaSim.log("Router " + name + " notifying its neighbors:");
    NovaSim.log(routerDV);
    for (int i = 0; i < links.size(); i++) {
        Link l = (Link) links.elementAt(i);
        Vector v = (Vector) routerDV.getEntries().clone();
        Message msg = new Message(v, name, l.getDestinationNode().getName());
        l.send(msg);
    }
}

/**
* @return a text representation of the router and its routing table
*/
public String toString()
{
    return "Router " + name + " routing table:
            
            " + routerDV.toString();
}

@Override
public void receive(Message msg)
{
    if (msg == null) {
        NovaSim.log("Router " + name + " received a null message!!!");
        return;
    }
}
Object msgObj = msg.getMessage();
String srcId = msg.getSource();

if (msgObj instanceof Vector) {
    /* Received neighbor's updated distance vector table */
    NovaSim.log("Router \" + name + \" received DV update message from node \" + srcId);
    receive((Vector) msgObj, srcId);
} else {
    /* Received normal message. Get the destination,
    * and either forward the message or handle it locally
    * /\n    NovaSim.log("Router \" + name + \" received a message from node \" + srcId);
    String destId = msg.getDestination();
    if (destId.equals(name)) {
        // Nothing to do. Log message and drop it
        NovaSim.log("Message \" + msgObj + \" from source \" + srcId + \" arrived successfully at router \" + name);
    } else {
        // TODO: reduce lifetime by one
        // TODO: implement getNextHop()
        String nexthop = "dummy"; // TEMP
        NovaSim.log("Router \" + name + \" forwarding received message to next hop \" + nexthop);
        forward(msg, destId);
    }

    private void forward(Message msg, String destId) {
        // TODO
    }
}
Appendix I – RouterEntry.java

/**
 * This class represents a routing-table entry.
 *
 * @author Anders Lindgren
 * @author Johan Nykvist
 */
public class RoutingEntry {
    /**
     * The destination router's ID number.
     */
    private String destination;
    /**
     * The distance (path cost) to the destination.
     */
    private int distance;
    /**
     * The next-hop router's ID number.
     */
    private String nexthop;

    /**
     * Constructs a routing entry with the specified arguments.
     *
     * @param destination the destination router's ID number
     * @param distance the distance to the destination
     * @param nexthop the next-hop router's ID number
     */
    public RoutingEntry(String destination, int distance, String nexthop) {
        this.destination = destination;
        this.distance = distance;
        this.nexthop = nexthop;
    }

    /**
     * Returns the destination router's ID number.
     *
     * @return the destination router's ID number
     */
    public String getDestination() {
        return destination;
    }

    /**
     * Sets the ID number of the destination router.
     *
     * @param destination the ID number of the destination router
     */
    public void setDestination(String destination) {
        this.destination = destination;
    }

    /**
     * Returns the distance (path cost) to the destination router.
     *
     * @return the distance to the destination router
     */
    public int getDistance() {
        return distance;
    }

    /**
     * Sets the distance to the destination router.
     */
}
public void setDistance(int distance) {
    this.distance = distance;
}

/**
 * Returns the next-hop router's ID number.
 * @return the next-hop router's ID number
 */
public String getNexthop() {
    return nexthop;
}

/**
 * Sets the ID number of the next-hop router.
 * @param nexthop the next-hop router's ID number.
 */
public void setNexthop(String nexthop) {
    this.nexthop = nexthop;
}

/**
 * Returns a text representation of the entry
 * @return a text string that represents the entry
 */
public String toString() {
    return destination + "\t" + distance + "\t" + nexthop;
}
import java.util.Vector;

/**
 * This class represents a routing table.
 *
 * @author Anders Lindgren
 * @author Johan Nykvist
 */
public class RoutingTable {

    /**
     * This Vector contains the RoutingEntry objects that
     * make the routing table
     */
    private Vector table;

    /**
     * Class constructor.
     */
    public RoutingTable() {
        table = new Vector();
    }

    /**
     * If needed, updates the entry for the specified destination with the
     * specified distance and the specified next-hop in the routing table. If no
     * entry for the specified destination exists in the table, a new entry is
     * created and added to the table. Returns <code>true</code> if changes
     * has been made to the routing table, otherwise <code>false</code>.
     *
     * @param destination
     *            the ID number of the destination router
     * @param distance
     *            the distance from this router to the destination
     * @param nexthop
     *            the ID number of the next-hop router
     * @return <code>true</code> if the routing table was updated, otherwise
     *         <code>false</code>
     */
    public boolean updateEntry(String destination, int distance, String nexthop) {
        RoutingEntry entry = getEntry(destination);
        RoutingEntry neighbor = getEntry(nexthop);
        if (neighbor == null) {
            NovaSim.showError("RoutingTable.updateEntry(): Can't find nexthop = " + nexthop + " from routing table");
            return false;
        }
        int newdistance = distance + getEntry(nexthop).getDistance();
        if (entry == null) {
            addEntry(destination, newdistance, nexthop);
            return true;
        }
        if (newdistance < entry.getDistance()) {
            entry.setDistance(newdistance);
            entry.setNexthop(nexthop);
            return true;
        }
        return false;
    }

    /**
     * Find destination from table
     */
    public RoutingEntry getEntry(String destination) {
        // Configuration problem...
        NovaSim.showError("RoutingTable.getEntry(): Can't find entry for destination = " + destination + " from routing table");
        return null;
    }

    /**
     * Find next-hop from table
     */
    public RoutingEntry getEntry(String nexthop) {
        // Configuration problem...
        NovaSim.showError("RoutingTable.getEntry(): Can't find nexthop = " + nexthop + " from routing table");
        return null;
    }

    /**
     * Add entry to table
     */
    public void addEntry(String destination, int distance, String nexthop) {
        RoutingEntry entry = new RoutingEntry(destination, distance, nexthop);
        table.add(entry);
    }

    public class RoutingEntry {

        /**
         * This class represents a routing entry in the routing table.
         *
         * @author Anders Lindgren
         * @author Johan Nykvist
         */
        private String destination;
        private int distance;
        private String nexthop;

        public RoutingEntry(String destination, int distance, String nexthop) {
            this.destination = destination;
            this.distance = distance;
            this.nexthop = nexthop;
        }

        public String getDestination() {
            return destination;
        }

        public int getDistance() {
            return distance;
        }

        public String getNextHop() {
            return nexthop;
        }

        public void setDestination(String destination) {
            this.destination = destination;
        }

        public void setDistance(int distance) {
            this.distance = distance;
        }

        public void setNextHop(String nexthop) {
            this.nexthop = nexthop;
        }
    }
}
* Adds a new entry for the specified destination with the specified metric
* and next-hop to the routing table.
*
* @param destination
* @param metric
* @param nexthop
* /
public void addEntry(String destination, int metric, String nexthop) {
    table.addElement(new RoutingEntry(destination, metric, nexthop));
}

/**
 * @return the routing table
 */
public Vector getEntries() {
    return table;
}

/**
 * Returns the RoutingEntry for the specified destination.
 * @param destination
 * @return the entry or <code>null</code> if no entry for the specified
 *         destination exists in the table
 */
private RoutingEntry getEntry(String destination) {
    for (int i = 0; i < table.size(); i++) {
        RoutingEntry re = (RoutingEntry) table.elementAt(i);
        if (destination.equals(re.getDestination())) {
            return re;
        }
    }
    return null;
}

/**
 * @return text representation of the routing table
 */
public String toString() {
    String r = "dest	dist	next hop
";
    for (int i = 0; i < table.size(); i++) {
        r += ((RoutingEntry) table.elementAt(i)).toString() + "\n";
    }
    return r;
}
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.util.Vector;  
import javax.swing.SwingUtilities;  
import javax.swing.Timer;  

/**  
 * @author Anders Lindgren  
 * @author Johan Nykvist  
 */  
public class Simulator {  
    private Vector eventQueue;  
    private double currentTime = 0;  
    private double step = 0.3;  
    
    /**  
     * Constructs a simulator instance.  
     */  
    public Simulator() {  
        eventQueue = new Vector();  
    }  
    
    /**  
     * Schedules the specified event to be invoked at the specified point in  
     * simulated time.  
     *  
     * @param event  
     *            the event to schedule  
     * @param time  
     *            the point in simulated time to invoke the event  
     */  
    public void schedule(Event event, double time) {  
        if (event == null || time < currentTime) {  
            System.err.println("Incorrect event!
");  
            System.exit(1);  
        }  
        event.setTime(time);  
        int i;  
        for (i = 0; i < eventQueue.size(); i++) {  
            if (((Event) eventQueue.elementAt(i)).getTime() > time)  
                break;  
        }  
        eventQueue.insertElementAt(event, i);  
    }  
    
    /**  
     * @param event  
     * @param time  
     */  
    public void reschedule(Event event, double time) {  
        unschedule(event);  
        schedule(event, time);  
    }  
    
    /**  
     * @param event  
     */  
    public void unschedule(Event event) {  
        eventQueue.removeElement(event);  
    }  
    
    /**  
     * @return the current simulated time  
     */  
    public double clock() {  
        return currentTime;  
    }  
}
return currentTime;
}

/**
* Starts the simulation. Stops when the event queue is empty.
*/
public void start() {
    Event e;
    while (eventQueue.size() > 0) {
        // step through events
        // step(step);
        e = (Event) eventQueue.elementAt(0);
        eventQueue.removeElementAt(0);
        currentTime = e.getTime();
        e.handle(currentTime);
    }
}

public void step(double steps) {
    Event e = (Event) eventQueue.elementAt(0);
    while (eventQueue.size() > 0 && (e.getTime() <= currentTime + steps)) {
        eventQueue.removeElementAt(0);
        currentTime = e.getTime();
        e.handle(currentTime);
        e = (Event) eventQueue.elementAt(0);
    }
}

/**
* Stops the simulation.
*/
public void stop() {
    System.out.println("Simulation stopped at " + currentTime);
    while (eventQueue.size() > 0)
        eventQueue.removeElementAt(0);
}

public double getStep() {
    return step;
}

public void setStep(double step) {
    this.step = step;
}
}
Appendix L – default_configuration.xml

<network>
  <node>
    <display><xcoord></xcoord><ycoord></ycoord>... </display>
    <name>1</name>
    <type>router</type>
    <interface>
      <id>192.168.0.101</id>
      <mask>255.255.255.0</mask>
    </interface>
    <interface>
      <id>10.0.1.11</id>
      <mask>255.255.255.128</mask>
    </interface>
    <interface>
      <id>222.111.0.1</id>
      <mask>255.255.255.192</mask>
    </interface>
    <interface>
      <id>111.0.222.1</id>
      <mask>255.255.254.0</mask>
    </interface>
    <interface>
      <id>100.200.10.21</id>
      <mask>255.255.252.0</mask>
    </interface>
  </node>
  <node>
    <display><xcoord></xcoord><ycoord></ycoord>... </display>
    <name>2</name>
    <type>router</type>
    <interface>
      <id>192.168.0.102</id>
      <mask>255.255.255.0</mask>
    </interface>
    <interface>
      <id>10.0.1.12</id>
      <mask>255.255.255.128</mask>
    </interface>
    <interface>
      <id>222.111.0.2</id>
      <mask>255.255.255.192</mask>
    </interface>
    <interface>
      <id>111.0.222.2</id>
      <mask>255.255.254.0</mask>
    </interface>
    <interface>
      <id>100.200.10.22</id>
      <mask>255.255.252.0</mask>
    </interface>
  </node>
  <node>
    <display><xcoord></xcoord><ycoord></ycoord>... </display>
    <name>3</name>
    <type>router</type>
    <interface>
      <id>192.168.0.103</id>
      <mask>255.255.255.0</mask>
    </interface>
    <interface>
      <id>10.0.1.13</id>
      <mask>255.255.255.128</mask>
    </interface>
    <interface>
      <id>222.111.0.3</id>
      <mask>255.255.255.192</mask>
    </interface>
    <interface>
      <id>111.0.222.3</id>
    </interface>
  </node>
</network>
<node>
  <display><xcoord></xcoord><ycoord></ycoord>....</display>
  <name>4</name>
  <type>router</type>
  <interface>
    <id>192.168.0.104</id>
    <mask>255.255.255.0</mask>
  </interface>
  <interface>
    <id>10.0.1.14</id>
    <mask>255.255.255.128</mask>
  </interface>
  <interface>
    <id>222.111.0.4</id>
    <mask>255.255.255.192</mask>
  </interface>
  <interface>
    <id>111.0.222.4</id>
    <mask>255.255.254.0</mask>
  </interface>
  <interface>
    <id>100.200.10.23</id>
    <mask>255.255.252.0</mask>
  </interface>
</node>

<node>
  <display><xcoord></xcoord><ycoord></ycoord>....</display>
  <name>5</name>
  <type>router</type>
  <interface>
    <id>192.168.0.105</id>
    <mask>255.255.255.0</mask>
  </interface>
  <interface>
    <id>10.0.1.15</id>
    <mask>255.255.255.128</mask>
  </interface>
  <interface>
    <id>222.111.0.5</id>
    <mask>255.255.255.192</mask>
  </interface>
  <interface>
    <id>111.0.222.5</id>
    <mask>255.255.254.0</mask>
  </interface>
  <interface>
    <id>100.200.10.24</id>
    <mask>255.255.252.0</mask>
  </interface>
</node>

<node>
  <display><xcoord></xcoord><ycoord></ycoord>....</display>
  <name>6</name>
  <type>router</type>
  <interface>
    <id>192.168.0.106</id>
    <mask>255.255.255.0</mask>
  </interface>
  <interface>
    <id>10.0.1.16</id>
    <mask>255.255.255.128</mask>
  </interface>
  <interface>
    <id>222.111.0.6</id>
    <mask>255.255.255.192</mask>
  </interface>
  <interface>
    <id>111.0.222.6</id>
    <mask>255.255.254.0</mask>
  </interface>
  <interface>
    <id>100.200.10.25</id>
    <mask>255.255.252.0</mask>
  </interface>
</node>
<node>
  <interface>
    <id>111.0.222.6</id>
    <mask>255.255.254.0</mask>
  </interface>
</node>

<node>
  <interface>
    <id>100.200.10.26</id>
    <mask>255.255.255.0</mask>
  </interface>
</node>

<link>
  <display>
    <xcoord></xcoord>
    <ycoord></ycoord>
  </display>
  <cost>10</cost>
  <from>192.168.0.101</from>
  <to>192.168.0.105</to>
</link>

<link>
  <display>
    <xcoord></xcoord>
    <ycoord></ycoord>
  </display>
  <cost>10</cost>
  <from>192.168.0.105</from>
  <to>192.168.0.101</to>
</link>

<link>
  <display>
    <xcoord></xcoord>
    <ycoord></ycoord>
  </display>
  <cost>6</cost>
  <from>222.111.0.1</from>
  <to>222.111.0.3</to>
</link>

<link>
  <display>
    <xcoord></xcoord>
    <ycoord></ycoord>
  </display>
  <cost>6</cost>
  <from>222.111.0.3</from>
  <to>222.111.0.1</to>
</link>

<link>
  <display>
    <xcoord></xcoord>
    <ycoord></ycoord>
  </display>
  <cost>13</cost>
  <from>222.111.0.2</from>
  <to>222.111.0.6</to>
</link>

<link>
  <display>
    <xcoord></xcoord>
    <ycoord></ycoord>
  </display>
  <cost>13</cost>
  <from>222.111.0.6</from>
  <to>222.111.0.2</to>
</link>

<link>
  <display>
    <xcoord></xcoord>
    <ycoord></ycoord>
  </display>
  <cost>11</cost>
  <from>111.0.222.2</from>
  <to>111.0.222.5</to>
</link>

<link>
  <display>
    <xcoord></xcoord>
    <ycoord></ycoord>
  </display>
  <cost>11</cost>
  <from>111.0.222.5</from>
  <to>111.0.222.2</to>
</link>

<link>
  <display>
    <xcoord></xcoord>
    <ycoord></ycoord>
  </display>
  <cost>14</cost>
  <from>100.200.10.22</from>
  <to>100.200.10.23</to>
</link>

<link>
  <display>
    <xcoord></xcoord>
    <ycoord></ycoord>
  </display>
  <cost>14</cost>
  <from>100.200.10.23</from>
  <to>100.200.10.22</to>
</link>
<link>
  <display><xcoord></xcoord><ycoord></ycoord>....</display>
  <cost>8</cost>
  <from>111.0.222.23</from>
  <to>111.0.222.24</to>
</link>

<link>
  <display><xcoord></xcoord><ycoord></ycoord>....</display>
  <cost>8</cost>
  <from>111.0.222.24</from>
  <to>111.0.222.23</to>
</link>

<link>
  <display><xcoord></xcoord><ycoord></ycoord>....</display>
  <cost>17</cost>
  <from>10.0.1.13</from>
  <to>10.0.1.15</to>
</link>

<link>
  <display><xcoord></xcoord><ycoord></ycoord>....</display>
  <cost>17</cost>
  <from>10.0.1.15</from>
  <to>10.0.1.13</to>
</link>

<link>
  <display><xcoord></xcoord><ycoord></ycoord>....</display>
  <cost>7</cost>
  <from>222.111.0.4</from>
  <to>222.111.0.5</to>
</link>

<link>
  <display><xcoord></xcoord><ycoord></ycoord>....</display>
  <cost>7</cost>
  <from>222.111.0.5</from>
  <to>222.111.0.4</to>
</link>

<link>
  <display><xcoord></xcoord><ycoord></ycoord>....</display>
  <cost>16</cost>
  <from>192.168.0.104</from>
  <to>192.168.0.106</to>
</link>

<link>
  <display><xcoord></xcoord><ycoord></ycoord>....</display>
  <cost>16</cost>
  <from>192.168.0.106</from>
  <to>192.168.0.104</to>
</link>

<link>
  <display><xcoord></xcoord><ycoord></ycoord>....</display>
  <cost>8</cost>
  <from>100.200.10.25</from>
  <to>100.200.10.26</to>
</link>

<link>
  <display><xcoord></xcoord><ycoord></ycoord>....</display>
  <cost>8</cost>
  <from>100.200.10.26</from>
  <to>100.200.10.25</to>
</link>

<error>
  <type></type>
  <time></time>
  <params></params>
</error>

</network>
Appendix M – output from the default configuration

[DEBUG] Adding router 1 and its interfaces
[DEBUG] Adding router 2 and its interfaces
[DEBUG] Adding router 3 and its interfaces
[DEBUG] Adding router 4 and its interfaces
[DEBUG] Adding router 5 and its interfaces
[DEBUG] Adding router 6 and its interfaces
[ERROR] Link starting from interface 100.200.10.23 already exist!

Simulation starts...

Router 1 notifying its neighbors:
dest  |  dist |  next hop
1     |   0   |    1
5     |   10  |    5
3     |    6  |    3

Router 2 notifying its neighbors:
dest  |  dist |  next hop
2     |   0   |    2
6     |   13  |    6
5     |   11  |    5
3     |   14  |    3

Router 3 notifying its neighbors:
dest  |  dist |  next hop
3     |   0   |    3
1     |   6   |    1
2     |   14  |    2
5     |   17  |    5

Router 4 notifying its neighbors:
dest  |  dist |  next hop
4     |   0   |    4
3     |   5   |    3
5     |   7   |    5
6     |   16  |    6

Router 5 notifying its neighbors:
dest  |  dist |  next hop
5     |   0   |    5
1     |   10  |    1
2     |   11  |    2
3     |   17  |    3
4     |   7   |    4
6     |   8   |    6

Router 6 notifying its neighbors:
dest  |  dist |  next hop
5     |   0   |    5
2     |   13  |    2
4     |   16  |    4
5     |   8   |    5

Router 2 received DV update message from node 6
Router 2 notifying its neighbors:
dest  |  dist |  next hop
2     |   0   |    2
6     |   13  |    6
5     |   11  |    5
3     |   14  |    3
4     |   29  |    6

Router 1 received DV update message from node 3
Router 1 notifying its neighbors:
dest  |  dist |  next hop
1     |   0   |    1
5     |   10  |    5
3     |    6  |    3
2     |   20  |    3

Router 4 received DV update message from node 6
Router 4 notifying its neighbors:
<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>29</td>
<td>6</td>
</tr>
</tbody>
</table>

Router 6 received DV update message from node 4
Router 6 notifying its neighbors:
<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>4</td>
</tr>
</tbody>
</table>

Router 5 received DV update message from node 4
Router 5 notifying its neighbors:
<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

Router 2 received DV update message from node 5
Router 2 notifying its neighbors:
<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>21</td>
<td>5</td>
</tr>
</tbody>
</table>

Router 5 received DV update message from node 2
Router 3 received DV update message from node 2
Router 3 notifying its neighbors:
<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>27</td>
<td>2</td>
</tr>
</tbody>
</table>

Router 3 received DV update message from node 1
Router 3 notifying its neighbors:
<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>27</td>
<td>2</td>
</tr>
</tbody>
</table>

Router 3 received DV update message from node 5
Router 3 notifying its neighbors:
<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>24</td>
<td>5</td>
</tr>
</tbody>
</table>
Router 3 received DV update message from node 1
Router 5 received DV update message from node 6
Router 6 received DV update message from node 5

Router 6 notifying its neighbors:

<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>5</td>
</tr>
</tbody>
</table>

Router 5 received DV update message from node 3
Router 3 received DV update message from node 2
Router 6 received DV update message from node 2
Router 5 received DV update message from node 4
Router 6 received DV update message from node 2
Router 2 received DV update message from node 3

Router 2 notifying its neighbors:

<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>3</td>
</tr>
</tbody>
</table>

Router 5 received DV update message from node 3
Router 2 received DV update message from node 3
Router 5 received DV update message from node 4
Router 5 notifying its neighbors:

<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

Router 6 received DV update message from node 2
Router 1 received DV update message from node 3

Router 1 notifying its neighbors:

<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
<td>3</td>
</tr>
</tbody>
</table>

Router 1 received DV update message from node 5
Router 1 notifying its neighbors:

<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>5</td>
</tr>
</tbody>
</table>

Router 5 received DV update message from node 4
Router 3 received DV update message from node 4
Router 2 received DV update message from node 3
Router 2 received DV update message from node 6
Router 1 received DV update message from node 5
Router 2 received DV update message from node 5
Router 1 received DV update message from node 3
Router 6 received DV update message from node 3
Router 6 notifying its neighbors:

<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>2</td>
</tr>
</tbody>
</table>
Simulation ended!

The stable routing solution is:

### Router 1

<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>5</td>
</tr>
</tbody>
</table>

### Router 2

<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>14</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>3</td>
</tr>
</tbody>
</table>

### Router 3

<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>24</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>23</td>
<td>6</td>
</tr>
</tbody>
</table>

### Router 4

<table>
<thead>
<tr>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Router 5</td>
<td>dest</td>
<td>dist</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Router 6</th>
<th>dest</th>
<th>dist</th>
<th>next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>18</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>