Programming Assignment 2: Distance Vector Routing

Due Date: 11 April 2018, 11:59 pm

Assessment: 30 marks

Goal: In this assignment your task is to implement the distance vector routing protocol. You will design a program that will be run at each router in the specified network. At each router the input to your program is a set of directly attached links and their costs. Note that the program at each router does not know the entire network topology. Your routing program at each router should report the cost and next hop for the shortest paths to all other routers in the network. You may extend the basic protocol by implementing a scheme to handle node failures and/or implementing poisoned reverse to solve the count-to-infinity problem.

Specification: You will implement the following program:

Distance Vector Routing (dv_routing)

- The program will accept the following command line arguments:
  - **NODE_ID**, the ID for this node (i.e. router). This argument must be a single uppercase alphabet (e.g.: A, B, etc).
  - **NODE_PORT**, the port number on which this node will send and receive packets from its neighbours.
  - **CONFIG.TXT**, this file will contain the costs to the neighbouring nodes. It will also contain the port number on which each neighbour is waiting for routing packets. An example of this file is provided below

- Since we can't let you play with real network routers, the routing programs for all the nodes in the simulated network will run on a single desktop/ laptop machine. However, each instance of the routing protocol (corresponding to each node in the network) will be listening on a different port number. If your routing software runs well on a single desktop machine, it will also likely to work on real network routers.

- Assume that the routing protocol is being instantiated for a node A, with two neighbours B and C. A simple example of how the routing program would be executed (assuming it is a Java program) follows:

  ```
  java dv_routing A 2000 config.txt
  ```

  where the config.txt would be as follows:

  ```
  2
  B 5 2001
  C 7 2002
  ```

- The first line of this file indicates the number of neighbours (not the total number of nodes in the network). Following this there is one line dedicated to each neighbour. It starts with the neighbour id, followed by the cost to reach this neighbour and finally the port number. For example, the second line in the config.txt above indicates that the cost to neighbour B is 5 and this neighbour is listening for routing packets on port number 2001. The node ids will be uppercase alphabets and you can assume that there will be no more than 10 nodes in the test scenarios, the link costs should be floating point numbers and the port numbers should be integers. These three fields will be separated by a single white space between two successive fields in each line of the configuration file. The link costs will be static and will not change once initialised. Further, the link costs will be consistent in both directions, i.e., if the cost from A to B is 5, then the link from B to A will also have a cost of 5. Also note that the nodes do not know the entire topology of the network, they only know the costs to their neighbours.

- Instead of implementing the exact distance vector routing protocol described in the textbook, you will implement a slight variation of the protocol. In this protocol, each node sends out the routing information (i.e. the distance vectors) to its neighbours at a certain frequency (once every 5 seconds), no matter if there have been any changes since the last announcement. This strategy improves the robustness of the protocol. For instance, a lost message will be automatically recovered by later messages.

- As specified in the distance vector protocols, your routing program at each node will exchange the distance vectors with directly connected neighbors. Real routing protocols use UDP for such exchanges. Hence, you MUST used UDP for exchanging routing information amongst the nodes. If you use TCP, you will not receive any marks for your assignment.

- It is possible that some nodes may start earlier than their neighbours. As a result, you might send the distance vector to a neighbour, which has not run yet. You should not worry about this since the routing program at each node will repeatedly send the distance vector to its neighbours and a slow-starting neighbour will eventually get the information.

- On receiving distance vectors from its neighbours, each node should re-compute its own distance vector. The format of the distance vectors exchanged between the neighbours should be similar to that discussed in the text (and the lecture notes). You should choose an appropriate format for these messages.

- Termination can also be a tricky part of your implementation. Real routing programs run forever without termination, but your program need to print out the routing table at some point in order to successfully complete the assignment. The key is to find out when the distance vectors have stabilized. We can assure you that when we test your program, we will start your routing program on all participating nodes within a short time period (e.g., 5 seconds) and there will not be link changes during the test. It will be your job to find out when the routing distance vector has stabilized and thus you can print out the output. Don't make us wait for more than three minutes!

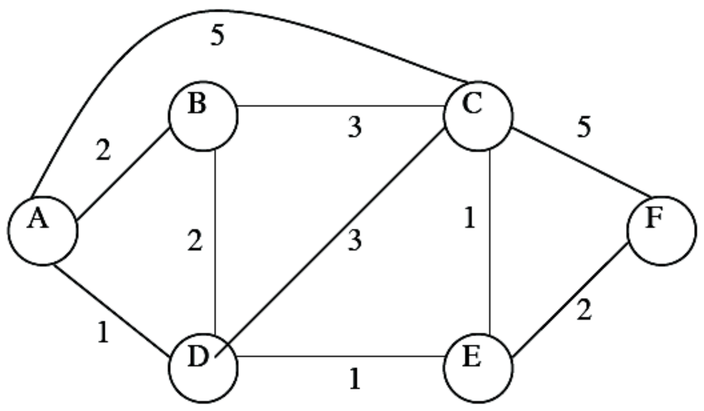
- Each node should print the final output to the terminal as shown in the following example (this is for node A in some arbitrary example):

```
Shortest path to node B: the next hop is D and the cost is 10
Shortest path to node C: the next hop is B and the cost is 11.5
```

- The routing protocol running on each node should continue to execute forever, exchanging distance vectors with its neighbours periodically (every 5 seconds). To kill an instance of the routing protocol, the user should type ***CTRL-C*** at the respective terminal.

An Example

- Lets look at an example with the network topology as shown in the figure below:



The numbers alongside the links indicate the link costs. The configuration files for the 6 nodes are available for download from the assignment webpage. In the configuration files we have assumed the following port assignments: A at 2000, B at 2001, C at 2002, D at 2003, E at 2004 and F at 2005. However note that some of these ports maybe in use by another student logged on to the same CSE machine as you. In this case, change the port assignments in all the configuration files appropriately. The program output at node A should look like the following:

```
shortest path to node B: the next hop is B and the cost is 2.0
shortest path to node C: the next hop is D and the cost is 3.0
shortest path to node D: the next hop is D and the cost is 1.0
shortest path to node E: the next hop is D and the cost is 2.0
shortest path to node F: the next hop is D and the cost is 4.0
```

Please ensure that before you submit, your program provides a similar output for the above topology. Of course you should not make any assumptions of the topology while coding. We will use a different network topology for marking.

Extensions: You may implement the following extension. The base assignment (i.e. the basic distance vector protocol described above) is worth 30 marks. The extensions is worth 6 marks. If you implement the extension as desired then you could potentially receive 6 bonus marks (i.e. a maximum of 36 marks). These marks will be added to the Practical Component of your marks. However, note that the marks for this component are capped at 100 marks (i.e. you cannot receive more than 100 marks on your practical component).

Extension: Handling Node Failure

- In this extension you must implement additional functionality in your code to deal with random node failures. Recall that in the base assignment specification it is assumed that once all nodes are up and running they will continue to be operational till the end when all nodes are terminated simultaneously. This extension will ensure that your algorithm is robust to node failures. Once a node fails its neighbours must quickly be able to detect this and the corresponding links to this failed node must be removed. Further, the routing protocol should converge and the failed nodes should be excluded from the shortest path computations.
- A simple method that is often used to detect node failures is the use of periodic heartbeat (also often known as keep alive) messages. A heartbeat message is a short control message, which is periodically sent by a node to its directly connected neighbours. If a node does not receive hearbeat messages from one of its neighbours it can assume that this node has failed. Note that each node transmits a distance vector to its immediate neighbour every 5 seconds. Hence, this distance vector message could also double up as the hearbeat message. Alternately, you may wish to make use of an explicit heartbeat message (over UDP), which is transmitted more frequently (i.e. with a period less than 5 seconds, say every second) to expedite the detection of a failed node. It is recommended that you wait till at least 3 consequent hearbeat (or distance vector) messages are not received from a neighbour before considering it to have failed. This will ensure that if at all a UDP packet is lost (UDP packet loss in a local network is very rare) then it does not hamper the operation of your protocol.
- Once a node detects that its neighbour has failed, it should recalculate its distance vector and exclude this neighbour in the computations. Further, the node need not compute the shortest path to this failed node. This newly computed distance vector should then be passed on to its other neighbours. Eventually, via the propagation of distance vectors, other nodes in the network will become aware that the failed node is unreachable and it will be excluded from all routing tables. Once a node has failed, you may assume that it cannot be initialised again.
- While marking your extension, we will only fail a few nodes, so that

a reasonable topology is still maintained. Further, care will be taken to ensure that the network does not get partitioned. In a typical topology (recall that the largest topology used for testing will consist of 10 nodes), at most 3 nodes will fail. However, note that the nodes do not have to fail simultaneously.

- As before, termination will be tricky, especially given that in this case the topology changes dynamically due to node failures. Once a node fails, its immediate neighbours will detect this and will re-compute their distance vectors. Eventually the distance vectors will not change further, following which your program should print out the routing table in the same format as before. Don't make us wait for more than three minutes after the node failure is initiated!
- When we test this extension, we will first start all nodes in the test topology and let the routing protocol stabilise and print out the output. Once the output is available at all nodes, the desired node(s) will be killed (by typing CTRL-C). We will again wait for your routing to stabilise and print the new shortest paths. If any further node failures need to be simulated, the above process will be repeated else all nodes will be terminated.

Programming Notes:

- Don't debug by embedding print statements in your code unless it is absolutely necessary. Instead, learn to use a symbolic debugger like gdb (or jdb for java). You can also find some nice GUI front-ends for gdb that make it easier to use. By using a debugger, you will save yourself tons of time finding the bugs in your code. Symbolic debuggers are used by professional programmers debug code so why not take advantage of this assignment to hone your professional skills. Note: Compile your code with the -g option in gcc so that the compiler will include debugging information in the executable. The javac compiler also has a -g option that does the same thing, but check the documentation.
- In writing your code, make sure to check for an error return from your system calls or method invocations, and display an appropriate message. In C this means checking and handling error return codes from your system calls. In Java, it means catching and handling IOExceptions.
- Make sure you close every socket that you use in your program. If you abort your program, the socket may still hang around and the next time you try and bind a new socket to the port ID you previously used (but never closed), you may get an error. Also, please be aware that port ID's, when bound to sockets, are system-wide values and thus other students may be using the port number you are trying to use. On Linux systems, you can run the command netstat to see which port numbers are currently assigned.
- Do not worry about the reliability of UDP in your assignment. It is possible for packets to be dropped, for example, but the chances of problems occurring in a local area network are fairly small. If it does happen on the rare occasion, that is fine. Further, your routing protocol is inherently robust against occasional losses since the

distance vectors are exchanged every 10 seconds. If your program appears to be losing or corrupting packets on a regular basis, then there is likely a fault in your program.

Additional Notes:

- This is not a group assignment. You are expected to work on this individually.
- While you are encouraged to adopt good programming practices, your coding style is NOT subject to marking.
- The programs will be tested on Linux machines. So please make sure that your entire application runs correctly on these machines. This is especially important if you plan to develop and test the programs on your personal computers (which may possibly use a different OS or version). Note that, the network will be simulated by running multiple instances of your program on the same machine with a different port number for each node. Make sure that your program will work appropriately under these conditions. See the sequence of operations listed below for details.
- **Error Condition:** Note that all the arguments supplied to the programs will be in the appropriate format. The configuration files supplied as an argument to each node will also be consistent with the test topology. Your programs do not have to handle errors in format, etc.
- **Language and Platform:** You are free to use either C, C++ or JAVA to implement this assignment (you can choose only one of these languages for the entire assignment, not a combination of them). Your assignment will be tested on the Linux Platform. Make sure you develop your code under Linux. As indicated earlier all nodes will be run on a single machine.

Assignment Submission:

- We will inform you about the details of submission in 2 weeks. Your routing program should be labeled appropriately to indicate the extensions that have been incorporated. You have the following 2 choices:
- dv_routing.c (or dv_routing.cpp or dv_routing.java) if you have only implemented the base case without any extensions.
- dv_routing_v1.c (or dv_routing_v1.cpp or dv_routing_v1.java) if you have implemented the extension (i.e. node failure).
- This naming convention is important since it will inform us what tests to run while marking your assignment.
- You may of course have additional header files and/or helper files. If you are using C/C++ you MUST submit a makefile. For JAVA users no makefile is required provided your program complies by using “javac *.java”.
- In addition you should submit a small report, report.pdf (no more than 2 pages) describing the design of your program. If you have attempted the extension briefly describe your approach. If your program does not work under any particular circumstances please report this here. Also indicate any segments of code that you have

borrowed from the Web or other books.

- **Late Submission Penalty:** Late penalty will be applied as follows:
  - 1 day after deadline: 10% reduction
  - 2 days after deadline: 20% reduction
  - 3 days after deadline: 30% reduction
  - 4 days after deadline: 40% reduction
  - 5 or more days late: NOT accepted

Plagiarism:

- You are to write all of the code for this assignment yourself. All source codes are subject to strict checks for plagiarism, via highly sophisticated plagiarism detection software. These checks may include comparison with available code from Internet sites. In addition, each submission will be checked against all other submissions of the current semester. Please note that we take this matter quite seriously. The Professor will decide on appropriate penalty for detected cases of plagiarism. The most likely penalty would be to reduce the assignment mark to ZERO. We are aware that a lot of learning takes place in student conversations, and don't wish to discourage those. However, it is important, for both those helping others and those being helped, not to provide/accept any programming language code in writing, as this is apt to be used exactly as is, and lead to plagiarism penalties for both the supplier and the copier of the codes. Write something on a piece of paper, by all means, but tear it up/take it away when the discussion is over. It is OK to borrow bits and pieces of code from sample socket code out on the Web and in books. You MUST however acknowledge the source of any borrowed code. This means providing a reference to a book or a URL when the code appears (as comments). Also indicate in your report, which portions of your code were borrowed and which were written yourself. Explain any modifications you have made (if any) to the borrowed code.

Sequence of Operation for Testing:

- The following shows the sequence of events that will be involved in marking your assignment. Please ensure that before you submit your code you thoroughly check that your code can execute these operations successfully.
  1. First chose an arbitrary network topology (similar to the test topology above). Work out the distance tables at each node using the methodology in the textbook (or lecture notes). Create the appropriate configuration files that need to be input to the nodes. Note again that the configuration files should only contain information about the neighbours and not of the entire topology.
  2. Log on to a Linux machine. Open as many terminal windows as the number of nodes in your test topology. Almost simultaneously, execute the routing protocol for each node (one node in each terminal).

```
java dv_routing A 2000 configA.txt (for JAVA)
java dv_routing B 2001 configB.txt
```

and so on.

3. Wait till the distance vector converges and the nodes display the output at their respective terminals.
4. Compare the displayed shortest paths to the ones obtained in step 1 above. These should be consistent.
5. For testing Extension, kill a few nodes to simulate node failures. As indicated in the specification, these should be carefully selected to avoid partitioning the network. Wait till the distance vector protocol converges and the nodes display the output at their respective terminals.
6. Terminate all nodes.

Marking Policy:

- For this assignment it is virtually impossible to delineate an exact marking policy. We will test your routing protocol for at least 2 different network topologies (of course different from the example provided). Marks will be deducted if necessary, depending on the extent of the errors observed in the output at each node.
- The distribution of the marks will be as follows:
  - Basic distance vector protocol: 30 marks.
  - Extension (handling node failures): 6 marks.

- **IMPORTANT NOTE:** We will not read through your code to evaluate your coding style, etc. For assignments that fail to execute all of the above tests after repeated attempts, we will be unable to award you a substantial mark. Such submissions will not receive more than 25% of the marks (depending on the correctness of the code).