COMP 3331/9331: Computer Networks & Applications

Programming Assignment 2 for Session 1, 2017

Due Date: 2nd June 2017, 11:59 pm (Fri, Week 13) ***Marks: 12 marks + 2 bonus***

Updates to the assignment, including any corrections and clarifications, will be posted on the subject website. Please make sure that you check the subject website regularly for updates.

1. Change Log

Version 1.0 released on 1st May 2017.

2. Goal and Learning Objectives

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. Your program must be able to elegantly handle the failure of neighbouring nodes. You must also implement poisoned reverse to solve the count-to-infinity problem.

精致的

2.1 Learning Objectives

On completing this assignment, you will gain sufficient expertise in the following skills:

- Designing a routing protocol
- Distance vector algorithm
- UDP socket programming
- Routing optimisations

3. Assignment Specifications

This section gives detailed specifications of the assignment. There are two versions of this assignment, a standard version (with a total of 12 marks) and an extended version (with a total of 14 marks of which 2 marks are bonus marks). The specifications for the extended version can be found in Section 5 of the specification. Note that the bonus marks may not be proportional to the amount of extra work that you will have to do. They are there to encourage you to go beyond the standard assignment. The bonus marks can be used to make up for lost marks in the lab exercises and the second programming assignment but NOT for any of the exams (mid-session and final).

3.1 Implementation Details

In this assignment, you will implement distance routing protocol.

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.
- POISONED REVERSE FLAG (-p), this is an optional argument, which is used to turn on/off Poisoned reverse. This means that your program should accept either 3 or 4 arguments. If the –p flag is present in the argument list then poisoned reverse should be employed in the routing protocol. If this flag is absent then only basic distance vector should be used.

Since we can't let you play with real routers, the routing programs for all the nodes in the simulated network will run on a single desktop 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 should also work on real network routers.

The file naming convention is explained later in the spec. For now, assume that the routing protocol is implemented in a file called *Dvr.java* (or *Dvr.c* or *Dvr.py*). The naming convention to be followed is explained later in the specification.

用具体例子运行说明

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

java Dvr A 2000 config.txt

$$\begin{tikzcd} & B \\ A \arrow[ur, dash] \arrow[dr, dash] & \\ & C \end{tikzcd}$$

where the config.txt would be as follows:
2 $\longrightarrow$ 2 neighbours
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. However, do not make assumptions that the node ids will necessarily start from the letter A or that they will always be in sequence. The link costs will be floating point numbers and the port numbers will 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. You may assume that the configuration files used for marking will be consistent with the above description and devoid of any errors.

Important: Each node is only aware of the costs to its direct neighbours. The nodes do not have global knowledge (i.e. information about the entire network topology).

To run your program with Poisoned reverse on, the following command should be used:

java Dvr A 2000 config.txt -p

Of course, consistency must be maintained across all nodes, i.e. Poisoned reverse should be either enabled or disabled globally.

Note: The rest of this discussion will outline the specification for the basic distance vector protocol without Poisoned reverse. A separate section later will discuss Poisoned reverse.

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 健壮 robustness of the protocol. For instance, a lost message will be automatically recovered by later messages. 通告

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 use **UDP** for exchanging routing information amongst the nodes. If you use TCP, a significant penalty will be assessed.

It is possible that some nodes may start earlier than their neighbours. As a result, a node 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 are free to choose an appropriate format for these messages.

Termination can be a tricky part of your implementation. Real routing programs run forever without termination, but your program must 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 stabilised. You can be assured that when we test your program, we will start the routing program on all participating nodes simultaneously using scripts and that the topology of the network will be stable during the test (except for the cases when we are testing for node failures and Poisoned reverse, as explained later). Your program should be able to determine when the routing distance vector has stabilized; following which it should print the output to terminal. **Don't make us wait for more than three minutes!**

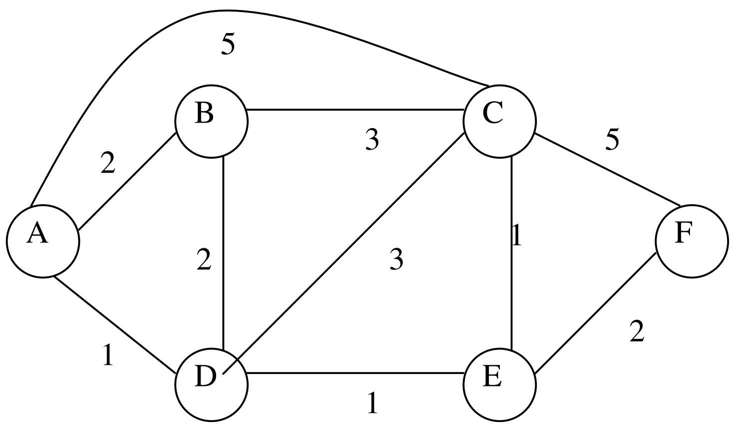
Each node should print the final output to the terminal as shown in the following example (this is for node A in some arbitrary example):

```
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.
各自的

3.2 An Example

Let us consider an example with the network topology as shown in the figure overleaf:



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. It may so happen that some of these ports may be in use because another student logged on to the same CSE machine as you and is running his/her program. You will receive an error message that indicates that the ports are in use. In this case, change the port assignments in all the configuration files appropriately and try again. 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
```

Note: It is not necessary that the nodes are listed alphabetically as in the example output above.

Before you submit, you should ensure that 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 set of different network topologies for marking.

3.3 Handling Node Failures

偶尔的

In the above description, 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. Occasionally, routers in the Internet may fail due to software or hardware problems. However, this does not adversely affect the flow of traffic. Other routers in the Internet are able to detect node failures and exclude the failed node from routing paths. Similar to real Internet routers, your routing protocol should be 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

明确的

neighbour before considering it to have failed. This will ensure that if at all a UDP packet is lost (though 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 from the vector computations. Further, the node need not compute the shortest path to this failed node. The 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 assignment, we will only fail a few nodes, such that a reasonable topology is still maintained. Further, care will be taken to ensure that the network does not get partitioned. However, note that the nodes do not have to fail simultaneously.

分割

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 the node failure feature, we will first start all nodes in the test topology and let the routing protocol stabilise and print the output. Once the output is available at all nodes, the desired node(s) will be killed (by typing CTRL-C in the respective terminal windows). 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 the remaining nodes will be terminated.

3.4 Implementing Poisoned Reverse

Your objective in this part of the assignment is to prevent the *count to infinity* problem that is known to affect the distance vector routing protocol by implementing *Poisoned Reverse*. (see the lecture notes on routing or the relevant sections on the text in Chapter 5)

As discussed earlier, your program should accept an additional optional parameter (-p), which should indicate whether the poisoned reverse module, is being used. This flag is used to turn on/off the poisoned reverse module. If the last flag is present in the argument list, then poisoned reversed is to be employed. The absence of this parameter indicates that only basic distance vector should be used.

In order to test poisoned reverse, we need to be able to change the link cost of certain links in the network topology. However, our basic configuration files do not provide this feature. Hence, we will use slightly different configuration files for testing this function. The following shows an example of the modified configuration file for node A:

```
2
B 5 50 2001
C 7 7 2002
```

As before, the first line indicates the # of neighbours. Notice the difference in the line representing each neighbour. In this case, the first letter indicates the node ID of the neighbour. This is followed by two link costs: the first number is for the initial link cost. The second number is the new link cost after the distance vector has converged at that node. In other words, once this node prints out the output to the terminal, it should assume that the link cost is now equal to the second number. As seen in the example above, the cost of the link from A to B is initially equal to 5. Once node A prints out

[illegible]

the shortest paths to all destinations, it should assume that the cost of the link to neighbour B changes from 5 to 50. However, notice that in the example above, the cost of the link to C does not change. To test this function, typically the cost of only one link in the entire network will be changed. You should also assume that the link cost changes in the configuration files for the two nodes at either ends of this link would be consistent. In the above example, the configuration file for node B will reflect the same change in the cost of the link from B to A. However, note that since the nodes may not print out their outputs at exactly the same time, they might detect the link change at slightly different instants of time. This is consistent with real network routers. Also note, that the modified configuration file as above will only be used when the –p flag is turned on.

Following the change in the link cost, your program will recompute the distance vector incorporating this change. However, you will need to be careful when this new vector is transmitted to the neighbours, because in poisoned reverse a node may be required to lie about its cost to certain destinations. Refer to routing lecture notes or the textbook to refresh your understanding of the principle behind poisoned reverse.

Your program must detect when the distance vector has converged following the change in the link costs and print out the required output to the terminal. In certain rare cases, depending upon the topology it may so happen that there is no change in a particular node's distance vector, i.e., it is completely unaffected by the change in the link cost. In this instance, this particular node does not have to do anything. It has already printed out its routing table to the prompt, which is still valid. Note that if poisoned reverse is not implemented it will take a very long time for the distance vectors to converge, especially if the change in the link cost is very large. On the contrary, in your implementation, the algorithm should converge very quickly. **Don't make us wait for more than three minutes to observe the output after the link change has taken effect!**

Configuration files for an example topology, which you can use to test your poisoned reverse implementation are available on the assignment webpage. This example is consistent with the example used in the lecture notes (and Figure 5.7 in the textbook). As before you may need to change the port numbers if you observe that the default ports used in these files are in use.

Note: If you are unable to implement Poisoned reverse, you may submit a scaled down version of the assignment, which lacks this functionality. This modification is subject to deduction in marks as indicated in the marking scheme. You should follow the file naming convention discussed later, if you do not implement Poisoned reverse.

4. Extension: Count-to-Infinity Issues (2 bonus marks)

NOTE: The amount of work involved in attempting the extension is not proportional to the bonus marks allocated. It is primarily geared towards high achieving students who want to take up the challenge.

You may implement the following extension if you wish to potentially receive **2 bonus marks**.

Even though poisoned reverse solves the count-to-infinity problem in some instances, it is not a generic fix for this problem. In other words, it will not solve all count-to-infinity problems. In this extension, your first task is to provide an example of a network where the count-to-infinity problem still occurs even if nodes employ poisoned reverse. You should provide this example in the report and clearly explain the existence of this problem.

Next, you should propose an enhancement that will solve the problem that you have described above. You must first describe your solution in your report. Following, this you must implement this extension in your program. Your extended program should accept an optional fifth argument (-e) for enabling your extension. You can use the same format of the configuration files as used for testing Poisoned reverse. In your report, you should provide a step-by-step explanation of how we can test your enhancement. You should also submit configuration files for an example topology, which we can use for these tests. You must describe this test topology in your report (perhaps you can use it to explain how your enhancement works). Note that, this enhancement is fairly open ended and you do not have to stick to the rigid structure of the basic assignment. You may also wish to tell us how we should execute your test topology.

5. Additional Notes

This is not a group assignment. You are expected to work on this **individually**.

How to start: Sample UDP client and server programs are available on the Week 3 lecture material page. They are a good starting point to start your development. You will also find several links to network programming resources on that page.

Backup and Versioning: We strongly recommend that you backup your programs frequently. CSE backups all user accounts nightly. If you are developing code on your personal machine, it is strongly recommended that you undertake daily backups. We also recommend to use a good versioning system so that you can roll back and recover from any inadvertent changes. There are many services available for both which are easy to use. We will NOT entertain any requests for special consideration due to issues related to computer failure, lost files, etc.

Language and Platform: You are free to use one of C, JAVA or Python to implement this assignment. Please choose a language that you are comfortable with. The programs will be tested on CSE 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 or JVM). Note that CSE machines support the following: **gcc version 4.9.2, Java 1.8, Python 2.7, 2.8 and 3.5. If you are using Python, please clearly mention in your report which version of Python we should use to test your code.** You may only use the basic socket programming APIs providing in your programming language of choice. 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.

You should 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 5 seconds. If your program appears to be losing or corrupting packets on a regular basis, then there is likely a fault in your program.

内在地.

6. File Naming Convention and Assignment Submission

You should use the following naming convention depending on the functionality that you have implemented:
DvrBase.c (or **DvrBase.java** or **DvrBase.py**) if you have only implemented the basic distance vector protocol without Poisoned Reverse.
DvrPr.c (or **DvrPr.java** or **DvrPr.py**) if you have implemented Poisoned Reverse on top of the basic distance vector protocol.
DvrExt.c (or **DvrExt.java** or **DvrExt.py**) if you have implemented the extension in addition to the above.

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 you MUST submit a makefile/script (not necessary with Java and Python). 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 extensions 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. If you attempted the extension, your report can be **4 pages** long.

You are required to submit your source code and `report.pdf`. You do not have to submit any topology files. The only exception is if you have attempted the enhancement, in which case you will need to provide for configuration files for a test topology. You can submit your assignment using the give command in a terminal from any CSE machine (or connecting via SSH to the CSE login servers). Make sure you are in the same directory as your code and report, and then do the following:

1. Type tar -cvf assign.tar filenames
e.g. `tar -cvf assign.tar *.java report.pdf`

2. When you are ready to submit, at the bash prompt type `3331`

3. Next, type: `give cs3331 assign2 assign.tar` (You should receive a message stating the result of your submission).

Note that, the system will only accept assign.tar as the file name. All other names will be rejected. You can submit as many times as you like before the deadline. A later submission will override the previous submission, so make sure you submit the correct version. Do not wait till just before the deadline for submission, as there may be unforeseen problems (brief disconnection of Internet connectivity, power outage, computer crash, etc.).

DO NOT SUBMIT ANYTHING ON OPENLEARNING.

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

NOTE: The above penalty is applied to your final total. For example, if you submit your assignment 1 day late and your score on the assignment is 10, then your final mark will be 10 – 1 (10% penalty) = 9.

7. 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 and assignments from previous semesters. In addition, each submission will be checked against all other submissions of the current semester. Do not post this assignment on forums where you can pay programmers to write code for you. We will be monitoring such forums. Please note that we take this matter quite seriously. The LIC will decide on appropriate penalty for detected cases of plagiarism. The most likely penalty would be to reduce the assignment mark to **ZERO**.

That said, we are aware that a lot of learning takes place in student conversations, and don't wish to discourage you from taking your classmates, provided you follow the Gilligan's Island Rule - After a joint discussion of an assignment or problem, each student should discard all written material and then go do something mind-numbing for half an hour. For example, go watch an episode of Gilligan's Island (or Reality TV in modern terms), and then recreate the solutions. The idea of this policy is to ensure that you fully understand the solutions or ideas that the group came up with.

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 the portions of your code that were borrowed. Explain any modifications you have made (if any) to the borrowed code.

8. Forum Use

Students are strongly recommended to discuss about the assignment on the course website. However, at no point should any code fragments be posted to the message forum. Such actions will be considered to be instances of plagiarism, thus incurring a significant penalty. Students are also encouraged to share example topologies that they have created to test their program.

9. 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 CSE 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 DvrBase A 2000 configA.txt (for JAVA)
java DvrBase 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) If you have attempted extension 1, let all nodes continue to run. Else kill all instances of the routing protocols.

6) For testing handling of failed nodes, 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.

7) Terminate all nodes.

8) If Poisoned Reverse has been implemented, the "-p" flag should be used to activate Poisoned reverse at all nodes. Select an appropriate topology and choose an appropriate link for which the link cost will be increased. Work out the distance vectors manually. Remember to use the modified configuration files for the nodes. Choose a large value for the link cost variation (typical cost variation should be 40 units or more). Following the convergence of the distance vector algorithm nodes will output the routing tables at the command prompt. Next, the change in the link cost will take affect and the distance vector computations will be performed again with poisoned reverse. Wait till the output is printed to the screen.

9) Terminate all nodes by typing CTRL-C in each terminal window.

NOTE: We will ensure that your programs are tested multiple times to account for any possible UDP segment losses (the occurrence of packet loss will be very rare).

10. Marking Policy

We will test your routing protocol for at least 2 different network topologies (which will be distinct from the example provided). Marks will be deducted if necessary, depending on the extent of the errors observed in the output at each node. After the marking process, we will upload the test topologies on the website for all students to view. Comments will be provided with each individual submission if marks have been deducted indicating the error in the outputs that we observed. The distribution of the marks will be as follows:

- Correct operation of the basic distance vector protocol: **6 marks**
- Correct handling node failures: **2 marks**
- Correct implementation of poisoned reverse: **3.5 marks**
- Report: **0.5 mark.**
- Extension: **2 marks (bonus).**