Level 11 Coursework — Distributed Systems 2012

Allan Clark

October 2012

1 Administration

Assigned Date:	Monday October 8th 2012
Due Date:	4pm Thursday November 22nd 2012

The expectation is that you should spend approximately 2-3 hours per week on practical work. You have a little over 6 weeks hence 12-16 hours work on this practical should be a reasonable amount of time, though of course this is subject to variability depending upon the student.

2 Part 1

2.1 Introduction

In this practical you will implement a simulation of a simple distributed algorithm. This will allow you to reason about a distributed algorithm prior to development and deployment on the potentially large scale of a real distributed system. You are to write a program which implements the Routing Information Protocol discussed in the networking fundamentals part of the course. This is a useful distributed algorithm which does not aim to have the exactly correct answer at all times but rather converge towards the current correct answer in response to changes. This can be adapted to many real world situations, for example content distribution.

Your program will need to take as input the description of the network, a set of addresses which each router node can reach directly and a set of initial events. The output of your program will be the set of events which follow, plus the resulting routing information table at each node of the network.

This is an individual practical, work may be discussed but all of the work you submit must be your own. You should submit a single source file containing your solution and answers to the questions at the end in comments.

The input of the program will be a network description, including the set of addresses which each router node can access directly and the links between each router node. Finally there will be a list of 'send' commands to initiate the algorithm. Frequently this will be a single command.

```
when a table is received by process p1 from process p2:
  for each row in the received table:
     if address is not known by p1:
        add the address to p1's table with the link "p2" and a
        cost of one more than the received cost
     if 1 + cost for the address is better than the current known one:
        place this row in p1's table with the link "p2" and a
        cost of one more than the received cost
     if address is known by p1 with a link of p2 then:
        if the cost for p2 is not exactly one less than p1's cost:
           act as if this address was unknown to p1
  if process p1 has updated its table in any way:
     send updated table to all links
```

Figure 1: Pseudo code for our version of the Routing Information Protocol.

The practical is designed such that your program will accept textual input and produce textual output. This means both that you are free to choose your implementation language and that I may automate evaluation of your solution to some extent. Whatever language you choose to implement your solution in you should make sure that it can be compiled and run on a standard DICE desktop.

2.2 Assumptions

We will assume that everything occurs within a 30 second time frame, hence your algorithm need not implement the periodic sending by each process of its Router Information Table. The pseudo code for the basic algorithm which we will follow is given in Figure 1.

We will further assume that no messages are dropped. Dropped messages are covered by the periodic sending of router tables and since we do not wish to model that we will assume that all messages eventually arrive at their destination.

We further assume that messages on the same link arrive in First-In-First-Out order, but only for messages on the same link, so if node `p1` sends node `p2` a message it will necessarily get there before any future message sent by node `p1` to node `p2`. However, node `p3` may subsequently send a message to node `p2` which arrives before the message from `node1`.

2.3 Input

Figure 2 shows a very simple input file. The input is a series of lines, one command per line with empty lines ignored. The network is described as a list of node commands and a list of link commands. Each node command

```
node p1 1
node p2 2
node p3 3
node p4 4 5

link p1 p2
link p1 p4
link p2 p3
link p3 p4

send p1
```

Figure 2: A simple example input file

is of the form `node name address*`, where `node` is a keyword, name is an alpha-numeric string giving the name of the node. Finally a list of integer addresses separated by white space, but on the same line. Link commands are of the form `link name1 name2`, specifying that there is a bi-directional link between nodes `name1` and `name2`. In some versions of the RIP algorithm links are given names which are referred to in the router information tables. Here we will always associate a table with a given node and hence the link will be entirely specified by the name of the destination node.

Finally in order to kick start the algorithm there will be at least one send command. This will specify less than a send event. It has the form `send name` where `send` is a keyword and `name` is the name of some node in the network. It simply states that the algorithm should be initiated by the given node sending its initial router information table to all of its links.

Finally your program may accept input either at `stdin` or as an input filename given as an argument on the command-line, please make clear which it is.

2.3.1 No parsing input

Since parsing textual input is not really the point of the practical it can be skipped. In this case put your input directly into the source code. Your code will be automatically tested however so please make sure that I can automate this. That format is akin to the textual input, so if you are doing this in Java the following is a good start:

```
private class InputNode {
  String name;
  int local_addresses;
  public InputNode (String n, int la[]){
    this.name = n;
    this.local_addresses = la;
  }
}
private class InputLink {
  String left_name;
  String right_name;
  public InputLink (String nl, String nr){
    this.left_name = nl;
    this.right_name = nl;
  }
}
private class InputCommand {
  String command_name;
  String process_name;
  public InputLink (String c, String p){
    this.command_name = c;
    this.process_name = p;
  }
}

private class Input{
  List<InputNode> input_nodes = new LinkedList<InputNode>();
  List<InputLink> input_links = new LinkedList<InputLink>();
  List<InputCommand> input_commands = new LinkedList<InputCommand>();
  public Input (){
    ... input statements ...
  }
}
```

To change the input, the line `... input statements ...` is changed to reflect the network, for example, our simple network would be added as:

```
  input_nodes.add(new InputNode("p1", {1}));
  input_nodes.add(new InputNode("p1", {2}));
  input_nodes.add(new InputNode("p1", {3}));
  input_nodes.add(new InputNode("p1", {4,5}));

  input_links.add(new InputLink("p1", "p2"));
  input_links.add(new InputLink("p1", "p2"));
  input_links.add(new InputLink("p1", "p2"));
  input_links.add(new InputLink("p1", "p2"));

  input_commands.add(new InputCommand("send", "p1");
```

You are of course then free to translate these simple types into your own classes and data representations for the running of the actual algorithm. The key point is merely that in order to test your program on a set of sample networks I should be able to copy and paste an input description of this form into your source code. If you are doing this in something other than Java please allow for an analogous set of commands and make clear what those commands are.

2.4 Output

The output of your program should be a list of events plus the final router information tables for all router nodes in the network. Each event is either a send or a receive, between two processes. A send event should be formatted as:

```
send p1 p2 (1|local|0) (2|p3|1)
```

More formally this is the keyword 'send' followed by two white-space separated node identifiers followed by the Router Information Table that is sent between the two. Router information tables are formatted as a list of rows, with each row in parentheses. A row is formatted as: `(address | link | cost)` Where `address` is an integer address, `link` is the process to which to send (for which there must be a link in the network) and cost is an integer stating how many links are required to get to the router which can send to the address directly. The `link` portion may also be the keyword `local` which indicates that the router to which the table is associated can deliver directly to the given address.

A receive command is given in exactly the same way but using the `receive` keyword:

```
receive p1 p2 (1|local|0) (2|p3|1)
```

Which means that process p2 receives the given table from process p1.

Figure 3 gives an example valid output file for a simple network. Note that although this is a valid output file it is not necsssarily a correct output file. Your output may be directed to a file or to `stdout`. Again please make clear which and also make sure that your program does not generate other data (such as debug statements) to the same place as the actual output. Though you can start a line-comment with a # character.

Note that in many descriptions or implementations of the Router Information Protocol the links are given names. Here we uniquely determine the link from the process to which the Router Information Table belongs and the process named as the link process. Also note that the initial 'send' commands specified in the input are not logged, but they trigger a 'send' event from one process to all the processes to which it is linked and those send events should be logged.

```
send p1 p2 (1|local|0) (2|p3|1)
receive p1 p2 (1|local|0) (2|p3|1)
table p1 (1|local|0) (2|p3|1)
table p2 (1|p1|1) (2|p3|2)
```

Figure 3: An example of valid (note not correct) output for a simple network.

2.5 Clarifications

- *Question*: Must I code my solution in Java?
- *Answer*: No, you may choose which ever programming language you prefer. However you should make sure that your solution can compile and run on a standard DICE machine. Additionally either you should allow text input, or provide a data-style input analogous to that of the Java version shown above. If you are in doubt please feel free to email me and ask.
- *Question*: Should I use threads in my solution?
- *Answer*: Threads is one way to simulate multiple communicating processes. But you are free to simply have a queue of events and an object for each process. You can then simply have a queue manager that takes events from the queue and calls the appropriate method on the object representing the process at which that event takes place. For example you could have a receive method on a process object which then queues further send events.
- *Question*: How should my simulator handle invalid network descriptions
- *Answer*: You needn't worry about validating or coping with invalid network descriptions. For example for every `link` command the two named nodes should always have associated `node` commands.

3 Part 2

In the second part of the practical you will extend your router information protocol simulator to handle events which break a link between two nodes. For this we must add an additional command to our input language, the `link-fail` command. This has the general form `link-fail name1 name2`, and means that the link between the nodes named `name1` and `name2` fails. The semantics of this is that all send commands in the input file are executed

```
receive p1 p2 (1|local|0) (2|p3|1)
link-fail p2 p1
send p2 p3 (1|no-link|i) (2|p3|1)
```

Figure 4: Shows part of a possible output of a network description which contains a link-fail command.

(or queued) at the beginning, but the link-fail commands are only executed (or queued) when the algorithm has first completed leaving the network in a stable condition. The link-fail command, should not be itself logged, but it should cause two events to be queued/reacted to and logged, these two events are: `link-fail name1 name2` and `link-fail name2 name1`. The first is received or executed at node `name1` and informs it that its link to node `name2` has been disabled. The second is the completely analogous event received at node `name2`.

When a process detects that a link has gone down, it should:

1. Record that it now no longer has a direct link to the named node
2. Update its routing information table such that all rows with the named node as the link have their cost set to infinity
3. Send updated table to all working links

We need a little extra syntax for the output of our router information tables. Suppose we have a table associated with node `p2` with a row that ends the initial algorithm as `(1|p1|1)`. Suppose the node `p2` then receives a `link-fail p2 p1` event. The row should be updated to reflect that it now has no route to address `1` and the cost for that is therefore ∞. We represent this as `(1|no-link|i)`

4 Loops in Routing

One shortcoming of the simple version of the router information protocol is that it is possible for there to be a temporary loop in the route for a given address. You should use your simulator to *help* find the conditions necessary for a loop to occur.

If you do not have time to complete the augmentation of your simulator, or for some reason it is not working, you may also try to work out by hand the conditions necessary to produce a cycle. You may not get full marks for your simulator but you can still attempt an answer to this question.

5 What to Submit

You must submit the source code of your solution. The source code should compile and run on a standard DICE desktop. This practical is not intended to evaluate your coding ability, however the portion(s) of your code which do the logic of the Router Information Protocol itself should be well marked and commented.

Your augmented simulator should work for both parts 1 and 2. In addition you should submit a description of the conditions necessary for a loop in routing to occur and how your simulator aided you in coming to this conclusion. Describe anything else you have added to your simulator in order to aid with this task.

Your answers to the additional questions should be given as comments in your source code. Please mark these clearly.

5.1 Evaluation

You will be evaluated primarily on how well your solution performs on sample networks. The key points are:

1. The resulting Router Information Tables should be correct.
2. No events should occur out of order, there is some scope for the order to be different as in a real network. However no event should violate the happens-before relation. In particular no process should receive a table before it is sent, nor send information gleaned from a table it has not yet received.
3. Your answers to the additional questions.

6 Grading

There are a total of 25 marks available for the entire practical. The implementation of your simulator for part 1 is worth a maximum of 17 marks, the augmentation to allow broken links a further 3. Finally your answer to the questions are worth a total of 5 marks.