Distributed Systems — Coursework

Allan Clark

School of Informatics
University of Edinburgh

http://www.inf.ed.ac.uk/teaching/courses/ds
Autumn Term 2012

Reminder

- Course work assigned Monday 8th of October
- Course work due:
  - Level 10 students: 4pm Thursday 8th of November
  - Level 11 students: 4pm Thursday 22nd of November

Coursework

- This year's practical will be mostly based upon a programming project
- Distributed algorithms are difficult to reason about and a simulator can provide excellent analysis of your algorithm whilst making explicit the assumptions we are making
- The main task is to implement a simulator for a distributed algorithm
- The algorithm in question is the Routing Information Protocol algorithm

Coursework

Reminder

- The Routing Information Protocol is a distributed algorithm to update the information router nodes have on where to forward packets to specific addresses
- It is not guaranteed to be constantly in the correct state, but it is known to converge on the correct state after an update to the network is made
- It is this convergence that we will simulate

Preliminaries

- The program you will write will be a command-line application
- It should accept text input and produce text output
- It can be written in any programming language you like within reason
- "Within reason" means:
  - Your program must compile and run on a standard DICE desktop
  - I must be able to read and understand your source code
  - Java, Haskell, Python, C, C++, Ocaml, SML — are all fine, though check your version is installed on DICE (or more rather your program compiles/runs on DICE)

Main Task

- Your program will ultimately accept a description of a router network
- Which describes the addresses each router node can dispatch to directly as well as the links between the router nodes
- Additionally you are told how to initiate the algorithm
- Your program should simulate the algorithm whilst logging each send and receive event
- The output will be the log of the send-receive events plus the final router information table for each router in the network

Reminder — Router Information Algorithm

- Each router node in the network maintains a table mapping addresses to:
  1. The link to which packets addressed to that address should be sent
  2. The cost associated with that route (for our purposes cost will be simply the number of hops)

-

Address	Link	Cost
1	local	0
2	p2	1
3	p2	2
4	p4	1
5	p4	1

Reminder — Router Information Algorithm

When a table is received by process p1 from process p2:

1. 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
2. if process p1 has updated its table in any way: send updated table to all links

Input

- Your program will accept a text input file which describes the network and an initial set of commands to kickstart the algorithm
- Each command is entered on a separate line
- The nodes are described by *node* commands:
  - `node p1 1 2 3`
  - `node` is a keyword, `p1` is the name of the node and the remainder is a list of integer addresses which are directly reachable from the given node.
- The links between nodes are described by `link` commands
  - `link p1 p2`
  - `link` is a keyword, `p1` and `p2` are node identifiers
- Finally kick start commands are given using the `send` command:
  - `send p1`
  - `send` is a keyword, this command means that the algorithm is kicked off by the node `p1` sending its routing information table (which consists of only the local nodes) to all links

Input Syntax

```
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
```

Initial State

The initial state of each node's router information table is therefore simply the list of local addresses:

- p1

1	local	0

- p2

2	local	0

- p3

3	local	0

- p4

4	local	0
5	local	0

No Parsing Input

Input without the Parsing

- Since this practical is not intended to test your ability to parse input data
- The network may be input as part of your source code
- This is described in the course work handout
- Essentially means that you must stick to an agreed set of standard type definitions — though you are then free to translate the input objects into your own objects
- In the coursework handout these are described for Java, if you are using a different language you should keep your definitions analogous — if in doubt ask!
- I personally think the parsing should only take around an hour and will help with your testing

Output Syntax

- Given an input network description your program should:
  1. Simulate the running of the router information protocol algorithm until there are no events left to execute
  2. Log each send and receive event and output this log
  3. Additionally output the final router information table for each router node

Output Syntax

Router Information Table Syntax

- You will need to format router information tables for both the output of the log of send-receive events and for the output of the final tables
- A table is formatted as a simple list of rows
- Each row is of the form: `(address|link|cost)`, including the parentheses
- Where, `address` is an integer, `link` is a node name *or* the keyword `local`, and `cost` is an integer.
- Note that some algorithms give names to links, but we will always associate a table with a node, so the link is simply the link between the node associated with the table and the node mentioned in the row.
- Example table for process p1: `(1|local|0) (2|p2|1)`

Output Syntax

- The output therefore begins with a log which consists of a set of `send` or `receive` commands
- `command name1 name2 rows*`
- Where `command` is either `send` or `receive`, `name1` is the origin of the message, `name2` is the destination of the message and `rows*` represents the rows of the table in the message
- `send p1 p2 (1|local|0) (2|p3|1)`
  - send from process p1 to process p2 the given table
- `receive p1 p2 (1|local|0) (2|p3|1)`
  - receive at process p2 the given table sent from process p1

Output Syntax

- At the end of the log of send-receive events should be the final tables, these are given by:
- `table name rows*`
- Where `table` is a keyword, `name` is the name of the router node associated with the table and `rows*` represent the rows of the table
- For example:
- `table p1 (1|local|0) (2|p3|1)`

Output Syntax

Example

```
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)
```

Note: there is no suggestion that this is the correct output for any particular network only that it is valid.

Output Syntax

- Please note that the syntax of both the input and output is fixed to allow me some level of automation when evaluating your solutions
- The easier it is for me to mark your solutions and provide feedback, the sooner I can return it to you

Level 10/11 Students

- Level 10 students must then answer a couple of questions based on their simulators
- The answers to these should be provided in the comments of your source code and clearly marked.

Level 10/11 Students

- Level 10 students must then answer a couple of questions based on their simulators
- The answers to these should be provided in the comments of your source code and clearly marked.

- Level 11 students face an additional task
- The simulator is to be augmented to allow for link breakages
- The input language is augmented with a command additional to the kickstarter `send` command
- A `link-fail` command specifies that a particular link should fail
- It has the form `link-fail p1 p2` where p1 and p2 are the names of the nodes which are connected by the link which should fail.

Level 11 Students Only

- `link-fail name1 name2`
- This should generate two new events in the log:
  1. `link-fail name1 name2` — in which the node `name1` detects that its link to `name2` has failed
  2. `link-fail name2 name1` — the node `name2` detects that its link to `name1` has failed.
- Your simulator should deal with the input `send` commands and wait for the algorithm to converge before then interpreting the `link-fail` commands.
- NOTE: There are two different things here, one is a command in the input file and the other is a log entry

Level 11 Students Only

Loops

- One of the weaknesses of the simple Router Information Protocol is that it is possible that the route for a given address is temporarily a cycle
- You will use this augmented simulator to attempt to find the conditions necessary for a temporary cycle in the routing information to occur.
- If your augmented simulator does not work, you can still submit an answer to this question, if you work out by hand the conditions necessary to produce a cycle.

Everybody Again!

Assumptions: in order to keep our task manageable we will keep to a few assumptions:

1. Everything happens within a 30 second window, so we do not need to implement the periodic sending of tables, this allows our algorithm to terminate
2. No dropped messages; messages may arrive out of order but they always eventually arrive, this allows us to avoid implementing periodic updates
3. Feel free to play with these assumptions but your final submission should turn those options off

Happens-Before again

Non-determinism

- Your simulator need not be deterministic
- In the sense that the same input need not necessarily produce the exact same output, you could introduce non-determinism by:
  - Use of threads
  - Use of a random-number-generator
- Neither does your program need to be non-deterministic
- Non-determinism could be used to decide which events get executed
- However, the important part is that none of your event logs can violate the *happens-before* relation
- If process p1 sends a message to process p2, and after that process p3 sends a message to p2 either may arrive first
- But if process p1 sends a message m, to process p2 then the send must occur before the receive and any send from p2 that relies on information from message m.

What to Submit

- A single source file containing your solution
  - I don't think this requires multiple source files
  - If you desperately disagree please "tar" up a directory containing all your source files
  - The clearer your source code is the easier it is for me to mark
  - However you are not judged on your coding style
  - The part(s) of your code that actually implements the logic of the router information protocol though should be clearly marked and documented.
  - Answers to the questions should be included in the source file as comments and clearly marked.

How long?

How long should this take?

- There are a wide range of students so an answer to this question is difficult and likely in accurate
- However, the expectation is that you spend 2-3 hours per week on coursework
- Level 10s you have just over four weeks and can therefore expect to spend around 8-12 hours
- Level 11s you have just over 6 weeks and can therefore expect to spend around 12-16 hours.

Yes I'm aware that $3 \times 6 = 18$

Distributed Systems Course Webpage

- The coursework description is available on the course website
- Both these slides and a "handout" giving more detail
- `www.inf.ed.ac.uk/teaching/courses/ds`
- Please check back regularly as there may be updates which would correspond to clarification on any issues raised

Marked Coursework

- We are committed to providing timely feedback on all coursework
- There are 126 students registered to this course
- There is one of me

Marked Coursework

- We are committed to providing timely feedback on all coursework
- There are 126 students registered to this course
- There is one of me
- alas: $\frac{126}{1} = 126$
- I aim therefore to return coursework within 3 weeks:
  - Level 10 students: Thursday November 29th
  - Level 11 students: Thursday December 13th
- But I hope to have a wrap-up lecture on the course work on Thursday November the 22nd dependent on the remainder of the course having been covered.

No lectures

Reminder:

- A final reminder that there are no lectures:
  - (this) Thursday the 11th of October
  - (week today) Monday the 15th of October
- Next lecture Thursday 18th October

Any Questions

Any Questions?

Any Questions

- Question: Can I use X programming language?
- Answer: Probably, provided your program compiles and runs on a standard DICE machine. There are many programming languages installed on DICE but be careful about which version you use if you plan on using some version specific feature. Also provided I am able to read your source code well.

Any Questions

- Question: Should I use threads in my solution?
- Answer: You can, but you do not need to.

Any Questions

- Question: Should I handle invalid network descriptions
- Answer: You needn't worry about input validation, for example if there is a link command then both of the named nodes shall have corresponding node commands. A node command may have no local links though.

Any Questions

- Question: What about a simulation that does not terminate
- Answer: That probably indicates a bug in your simulator or your RIP algorithm implementation. It would be considerably good form if you either figure out how to detect this or come up with an upper bound on the number of events that should occur before the algorithm terminates and hence exit-fail after that many events.