COSC 331
Data Communications and Networks
Assignment

Andreas Willig
Dept. of Computer Science and Software Engineering
University of Canterbury
email: andreas.willig@canterbury.ac.nz

July 25, 2011

---

1 Administrivia

This assignment is part of the COSC 331 assessment process. It is worth 20 marks, i.e. 20% of the final 100 marks. It is a programming project in which you implement parts of the RIP protocol [1] and run a single instance of it in a single process under the Linux operating system. The intention is to start several of these processes on the same machine and to connect them through sockets, not through real networks. You can work in groups of two students.

You can use Python, Java or C, and you have to use the socket interface implementation that is available for your programming language with its standard libraries. Your program has to be executable under Linux (more precisely: it should run on the Fedora Core 14 machines running in Lab 1). Your program should be written so that it does not make use of any facilities (libraries etc.) that are not part of the standard distribution of your chosen programming language and cannot be easily distributed with the deliverable package (see Section 3). The marker will not install separate software or to apply special configurations to test your programs.

It is quite likely that the problem description below leaves a lot of things unclear to you. Please do not hesitate to use the "Question and Answer Forum" on the Learn platform for raising and discussing any unclear issues.

2 Problem Description

You will implement a "routing demon" as a normal userspace program under Linux. Instead of sending its routing packets over real network interfaces, your routing demon will communicate with its peers (which run in parallel on the same machine) through local sockets. Your program should be a text mode program, no extra credit will be given for providing a graphical user interface.

Roughly speaking, your program will operate in three stages:

- At the beginning it reads a configuration file (whose name is supplied as a command line parameter) which contains a unique identification for the routing demon instance, the port numbers on which the demon receives routing packets from peer demons (**input ports**), and the port numbers to which to send own routing packets (**output ports**). Clearly, any output port of one router should be an input port of another router.

- Next the demon creates as many sockets as it has input ports and binds one socket to each input port.

- Finally, you will enter an infinite loop in which you react to incoming events (check out the `select()` system call or its equivalent in your favorite programming language). An incoming event is either a routing packet received from a peer (upon which you might need to update your own routing table, print it on the screen, or possibly send own routing messages to your peers), or it is a timer event upon which you send an unsolicited RIP response message to your peers. Please ensure that you handle each event atomically, i.e. the processing of one event must not be interrupted by processing another event.

All the routing demons will be instantiated on the same Linux machine, so you will have to use the IP address for the local host. You should use the UDP protocol for routing messages.

One hint for your design: many protocol specifications (and even more so the implementations) are expressed using the formalism of extended finite state machines or finite automata (i.e. state machines / automata that have variables added and where transitions can be conditioned on both an event and the value of one or more variables). This can be a very useful design framework. You would have to identify all possible events, the set of variables, and the set of states. You then have to specify for each state what will happen when when an event comes in. This has to be done for each event.

2.1 The Configuration File

Your program should be a single executable which can be started with a single command line parameter. This parameter will be the filename of a configuration file. The configuration file should allow users to set at least the following parameters (one parameter per line, please allow for empty lines and comments):

- Router id, which is the unique id of this router. A suggested format[1] for this parameter is:

  ```
  router-id 1
  ```

  The router id is a positive integer between 1 and 64000. Each router must have its own unique router-id, and thus each router configuration file must have a different value for this parameter.

- Input port numbers: this is the set of port numbers (and underlying sockets) to which the started instance of the routing demon will listen for incoming routing packets from peer routing demons. A suggested format for this parameter is:

  ```
  input-ports 6110, 6201, 7345
  ```

  i.e. the port numbers could be separated by commas. You do not have to follow precisely this format, but your parser for the configuration should ensure that:
  - All port numbers are positive integers x with $1024 \leq x \leq 64000$
  - You can specify arbitrarily many of them, but all in one line
  - Each value occurs at most once.

- Output port numbers: here you specify the port numbers of "neighboured" peer routers with which you exchange routing information. The numbers must be distinct from those numbers listed with the `input-ports` parameter. A suggested format for this parameter is:

  ```
  output-ports 5000-1, 5002-5
  ```

  A single entry (like `5000-1`) consists of a port number and a metric value, separated by a hyphen. The port numbers given here must satisfy the same conditions as the port numbers given for the `input-ports` parameter. Furthermore, none of the port numbers given for the `output-ports` parameter should be listed for the `input-ports` parameter. The metric values should conform to the conditions given in [1].

- Values for the timers you are using in your implementation. It is wise to use timer values (for periodic updates and timeouts) smaller than the ones given in the standard, since this can shorten experimentation times. However, please ensure that the ratio of the timer values remains the same ($180/30 = 6$ for timeout vs. periodic, similarly for garbage collection).

[1] This is really just a suggestion. Feel free to choose the format so that writing the parser becomes as easy as possible for you, but please ensure that your format is human-readable (i.e. no binary format, syntax not too cluttered).

The first three parameters (`router-id`, `input-ports`, `output-ports`) are mandatory, if one of them is missing your program should stop with an error message. When you set up several configuration files for several routers, you must ensure manually that the following conditions are met:

- The `router-id`'s of all routers are distinct
- When you want two routers A and B to be neighbored, you should:
  - provide an input port x at B that is also listed as output port of A
  - provide an input port y at A that is also listed as output port of B
  - ensure no other host than A has listed port x as an output port
  - ensure no other host than B has listed port y as an output port
  - and finally, ensure that the metrics that A specifies for its output port x is the same as the metric that B specifies for its output port y.

2.2 Exchange Routing Packets and Route Computations

In this section it is assumed that you are already intimately familiar with the RIP protocol specification [1]. Here we will just give a few hints on which parts of the specification can be ignored, should be simplified, or need to be modified.

- Scope of implementation is essentially Section 3 of [1]. None of the extensions in section 4 is to be included (no extra credit). Please implement split-horizon with poisoned reverse.
- Implement triggered updates only for deleted routes, not for metric updates or new routes.
- Do not use the regular UDP port, but the port numbers from the configuration file.
- Do not implement any retransmission scheme for request messages.
- The version number is always 2.
- You do not route to subnetworks and do not handle IPv4 addresses, but you only route to the various routers (as identified by their `route-id`). You also do not need to take care of default addresses, host routes or subnet masks.
- RIP request packets are always whole-table requests and a RIP router sends them on all its output ports.
- RIP response packets are always sent to all output ports.
- Do not multicast request messages, just send them to the intended peer through its socket.

- Packet format:
  - ignore the issue of network byte order
  - Please use the 16-bit wide all-zero field in the RIP packet common header for the `router-id` (since otherwise you would have no identification of the router sending out an update. Furthermore, you should also work with `router-id`'s instead of IPv4 addresses. Represent `router-id`'s internally as 32 bit numbers.
  - please perform consistency checks on incoming packets: have fixed fields the right values? Is the metric in the right range? Non-conforming packets should be dropped (and perhaps a message printed, this helps your debugging).
- Regarding periodic updates: You need to find out how to generate periodic timer events and how to write own handlers for this event. This depends on your choice of programming language. It could also be useful to introduce some randomness to the periodic updated, for example by setting the timer to the given period value the value of a random variable that is uniformly distributed over the range $[-0.2 \cdot \text{period}, 0.2 \cdot \text{period}]$. Why could such a randomization be useful?

Some other things:

- If a demon has several input ports, it needs to listen to all of them simultaneously. This can be achieved through the `select()` system call or its equivalent in your chosen programming language.

It goes without saying that you should perform various tests with your setup. These tests should show that your routing protocol design and implementation correspond "in the right way" to certain failure events (e.g. one process is shut down and re-started later). To properly conduct these tests it is also important to formulate **in advance** which behaviour your implementation **should** show and to compare the actual outcome against it.

3 Deliverables

Each student has to submit the following items:

- a `.tgz` archive (`tar`'ed and `gzip`'ed, check out the man page for `tar`) including all your source code and configuration files. Remember that the program must be able to run out of the box, without needing us to install any additional software. Please also include a `README` file which briefly explains how to run your program.
- the configuration files should allow to set up the same network of routers as has been used on the routing problem sheet.
- a pdf file giving your responses to **all** questions given below. Please **do not use other formats**.

These items have to be submitted to the departmental coursework dropbox, see `http://cdb.cosc.canterbury.ac.nz`. Please submit no later than **Monday, September 12, 2011, 5pm**. Late submissions are accepted until one week later but will receive a 15% absolute penalty (i.e. you will loose 3 out of 20 marks).

4 Questions and Marking

Please answer each question with one or two paragraphs at most.

- Who was your partner?
- What have you contributed to the design?
- What has your partner contributed to the design?
- What have you contributed to implementation and testing?
- What has your partner contributed to implementation and testing?
- How would you rate your overall contribution to the project as a percentage?
- Which aspects of your overall program (design or implementation) do you consider particularly well done? Please answer the same question for **your contribution**.
- Which aspects of your overall program (design or implementation) could be improved? Please answer the same question for **your contribution**.
- How have you ensured atomicity of event processing?
- Which tests have you performed and why do these tests establish that your design and implementation operate correctly? Please explain (you might need more than two paragraphs).

Marking will be based on these answers and on the delivered software. There will be deductions if the software does not run properly or if the source code is poorly written.

References

[1] G. Malkin. RIP Version 2. *RFC 2453*, 1998.