COSC 364 Internet Technologies and Engineering RIP Assignment Description

Andreas Willig
Dept. of Computer Science and Software Engineering
University of Canterbury

February 13, 2025

1 Administrivia

This assignment is part of the COSC 364 assessment process. It is a programming project in which you implement parts of the RIP routing protocol [1] and run several instances of a RIP daemon under the Linux operating system on the same machine (no virtual machines involved!). Each instance runs as a separate process, and these processes communicate through local sockets. You will emulate a small network and explore the response of the RIP protocol to different types of faults.

You will have to use the Python3 programming language, and you have to use the socket interface implementation that is available for Python. **Important:** Your program has to be executable under Linux from the `bash` command line!

It is quite likely that the problem description below leaves a lot of things unclear to you. Do not hesitate to use the "General discussion forum" on the LEARN platform for raising and discussing any unclear issues. **Important:** Please do not email the lecturer with technical questions, rather send such questions to the LEARN forum, so that all students can benefit.

Important: Before reading the following problem description in more detail, it is absolutely essential that you read the RIP specification [1], which you also find on the LEARN page of the course. The scope of this project is essentially given by Section 3 of [1], none of the extensions in Section 4 is to be included (if you do nonetheless, no credit will be given).

2 Pair Work

An important sub-goal of this assignment to give you some pair-work experience. The rules around this are as follows:

- You are expected to work in groups of two persons, submissions of larger groups will not be marked.
- It is your responsibility to find a partner.
- An individual submission comes with an automatic penalty of 15%, unless there is exactly one individual submission. No exception other than those related to "special consideration" will be given to this rule. Clearly, sometimes things may not work out well with a partner. An important piece of advice in this context is to **start early** with the assignment, so that you have enough time to find a new partner, should that happen.

If you do not find a partner on your own (you could try to post on the LEARN forum!), then you can send an email to the lecturer. The lecturer will then try to find a suitable partner for you from those who sent similar emails.

3 Plagiarism Warning

Your submissions are logged and originality detection software will be used to compare your solution with other solutions (from this year and from previous years). Dishonest practice, which includes

- letting someone else create all or part of an item of work,
- copying all or part of an item of work from another person with or without modification,
- seeking out or viewing submissions of others, and
- allowing someone else to copy all or part of an item of work,

will lead to full loss of assignment marks, failing the course, or other serious consequences including notification of the University Proctor.

Anything you submit for credit must be entirely your own work and not adapted or copied, with or without modification, from any other person. **Your source code and text must be completely your own**, any sharing of source code with other groups / persons or any viewing of someone else's source code is expressly **forbidden**. If you need help with specific details relating to your work, or are not sure what you are allowed to do, contact your tutor or lecturer for advice. If you copy someone else's work or share details of your work with anybody else, you are likely to be in breach of university regulations and the Computer Science and Software Engineering department's policy. For further information please see

- Academic Integrity Guidance for Staff and Students
  `www.canterbury.ac.nz/ucpolicy/GetPolicy.aspx?file=Academic-Integrity-Guidance-For-Staff-And-Students.pdf`
- Academic Integrity and Breach of Instruction Regulations in the University Calendar
  `www.canterbury.ac.nz/regulations/general-regulations/academic-integrity-and-breach-of-instruction-regulations/`

4 Problem Description

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

Roughly speaking, your daemon program will operate in three stages:

- At the beginning it reads a configuration file (whose name has to be supplied as a command line parameter – please avoid any interactive reading of the filename) which contains a unique identification for the routing daemon instance, the port numbers on which the daemon receives routing packets from peer daemons (**input ports**), and specifications of the **outputs** towards neighbored routers. Clearly, any output port declared for one router should be an input port of another router. The information in the configuration file is only meant to inform daemons about links, the daemons internal routing table must not be initialized from the configuration file.
- Next the daemon creates as many UDP sockets as it has input ports and binds one socket to each input port – no sockets are created for outputs, these only refer to input ports of neighbored routers. One of the input sockets can be used for sending UDP datagrams to neighbors.
- Finally, you will enter an infinite loop in which you react to incoming events (check out the `select()` system call and its equivalent in Python.[1]) 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 (e.g. a received packet) must not be interrupted by processing another event (e.g. a timer).

[1] One of `select()`'s important properties is that it *blocks* while you wait for events, i.e. your program does not consume CPU time. This property is important and your program needs to have it. Note that `select()` blocks an entire process – if your process contains several threads (e.g. one per router) then *all* threads are blocked, not only the calling one.

All the routing daemons will be instantiated on the same Linux machine, so you will have to use the IP address for the local host (`127.0.0.1`). You should use the UDP protocol for routing messages.

One hint for your design: many protocol specifications (and even more so their 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.

4.1 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 be an ASCII file that can be **read and edited by humans**. It is recommend to choose a very simple syntax for the configuration file and to keep the parser very easy, no extra credits will be given for an elaborate syntax (nothing XML-ish or any other similarly baroque format!), fancy parsing code or the like. However, basic syntax and consistency checks (when a number is expected it should be checked that indeed a number is present and is in the allowed range) **need to be made**. Each instance of the router daemon will be started with a separate 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 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) on which the instance of the routing daemon will listen for incoming routing packets from peer routing daemons. There needs to be a separate input port for each neighbor the router has. 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 $1024 \leq x \leq 64000$ (find out why 1024 was chosen as lower bound!)

  - You can specify arbitrarily many such entries, but all in one line
  - Each port number occurs at most once.

- Outputs: here you specify the "contact information" for neighboured routers, i.e. routers to which a direct link should exist and with which you exchange routing information. A suggested format for this parameter is:

  ```
  outputs 5000-1-1, 5002-5-4
  ```

  As before, different outputs are separated by commas. For one output, for example `5002-5-4`, the first number `5002` specifies the input port number of the peer router, i.e. the port number on which the peer router will listen to packets from the current instance. The second number `5` represents the metric value for the link to the peer router. The third number `4` represents the router-id of the peer router, so that your instance knows which router is really behind this link. 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 `outputs` 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).

The first three parameters (`router-id`, `input-ports`, `outputs`) are mandatory, if one of them is missing your program should stop with an error message. When you set up several configuration files (one for each router), 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
  - ensure no other host than A has listed port y as an input port
  - ensure no other host than B has listed port x as an input
  - and finally, ensure that the metrics that A specifies for its output with port number x is the same as the metric that B specifies for its output port y.

4.2 Exchanging 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. This means that for any update message or unsolicited response a separate packet needs to be created for each neighbor.
- Implement triggered updates only when routes become invalid (i.e. when a router sets the routes metric to 16 for whatever reason, compare end of page 24 and beginning of page 25 in [1]), not for other metric updates or new routes.
- Do not use the UDP port specified in [1], but the port numbers from the configuration file.
- Do not implement request messages, you should use only the periodic and triggered updates.
- 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 `router-id`). You also do not need to take care of default addresses, host routes or subnet masks.
- RIP response packets always include the entire routing table, and to each neighbor a separate copy of this table is sent (according to the split-horizon-with poisoned reverse rule, which implies that each neighbor might get a different view of the table).
- Do not multicast response messages, just send them to the intended peer through its input port.
- Please use the RIP Packet format with the following changes:
  - 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.
  - **Important:** 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).

In Python, packets should be constructed as byte-arrays.

- Regarding periodic updates: You need to find out how to generate periodic timer events and how to write own handlers for this event. It could also be useful to introduce some randomness to the periodic updates, for example by setting the timer in question randomly to a value that is uniformly distributed over the range $[0.8 \cdot \text{period}, 1.2 \cdot \text{period}]$. Why could such a randomization be useful?

Some other things:

- If a daemon has several input ports, it needs to listen to all of them simultaneously. This can be achieved through the `select()` system call / its Python equivalent.

It goes without saying that you should perform various tests with your setup. These tests should show that your routing protocol design and implementation converge to the "right" routing tables (which you will have to establish from inspection) and respond "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. The effort spent on testing and its documentation in the report will play a substantial role in marking the report.

5 Deliverables

To receive marks for the assignment, each pair of students has to do two things:

- demonstrate your program and explain the source code to me during an in-person inspection.
- Submit a **`.pdf` file with all fonts embedded**, which includes:
  - A cover sheet with your names and student-id's.
  - Answers to questions (see below).
  - One example configuration file used in the demonstration.
  - Your complete Python source code.

Please make sure only one member of a team submits the report.

Warning:

- Reports that are not in pdf format or which contain more than one file automatically receive 0 marks!!
- Please make sure that your `.pdf` file contains all fonts and can be completely printed on UC printers – if it cannot be printed it will **not be marked**.
- By submitting, you declare that you have read and understood the plagiarism warning in this assignment description and that your submitted work is entirely yours. Should we find evidence that you have engaged in plagiarism-related activities, you will automatically receive 0 marks for the assignment, and your name will be submitted to the University Discipline Register!!

5.1 The Inspection / demonstration

The demonstrations will take place after assignment report submission, dates will be arranged around the submission date. A demonstration session will use the Linux computers in the COSC labs (**Important**: make sure your program runs from the `bash` command line under Linux!) and will have the following structure:

- The first ten minutes are a live presentation of the program:
  - First we will start up all router instances and check whether the routing tables converge and the resulting routes are "minimum hop". Please make sure that your program generates sufficient output for the lecturer to check this – for example, you could print the routing table after each change or each periodic update. **Important note:** Please do not print anything else, like debug output. If you do, or if you do not print routing tables frequently in the first place, the lecturer will ask you to make changes on the spot and apply deductions.
  - Next, we will switch off one or more of the routers and expect that the remaining network converges again into a minimum hop state after a flurry of activity.
  - Finally we will switch the failed router(s) on again and expect to see that the network converges back into the initial state.
- The second ten to fifteen minutes are devoted to a code walk-through of your program, to be done by the first partner. The second partner of a team will wait outside. **Important**: each partner should be able to explain the entire program!
- The other partner will do the code walk-through in the third ten to fifteen minutes.

The most important factors influencing the marking are:

- The achieved functionality as shown in the first ten minutes.
- The contribution percentage of either partner.
- Your understanding of the code and your ability to explain it in detail (as shown in the walk-through).

The functionality counts most, the source code as such does not count at all, except that you can lose marks should it be particularly ugly or unreadable.

For the demonstration you need to set up configuration files for the example network shown in Figure 1. In the figure, the vertices represent routers and their router-id, the edges represent bi-directional links. The edges are labeled with cost values for the respective link.

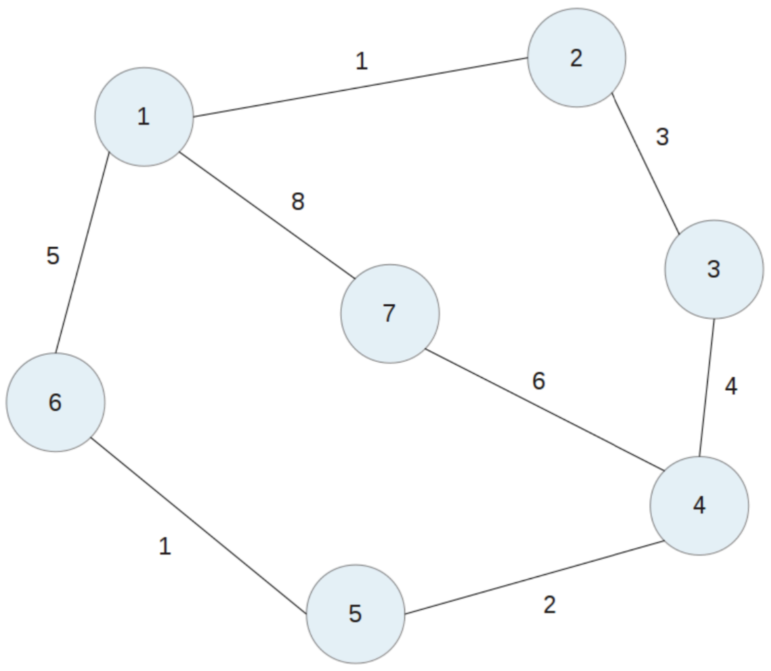

Figure 1: Example network for demonstration

Please make sure that we can see what is happening during the demonstration, and that everything is ready to run when we meet. In particular, each router instance should print its current routing table periodically and/or after each change in an easily readable format. The table should list all the destinations, the costs, the next hop router, and the current values of the timers associated with the destination. **Do not** print any other output, e.g. debug output.

5.2 The Report

The report needs to be a **single `.pdf` file** for each pair. The report needs to include:

- A title page listing name and student-id of both partners.
- A percentage contribution of each partner to the overall project. **Important**: this must be **agreed upon** by you and your partner, the relative weights will influence grading.
- For each partner please give a brief list of contributions.
- There should be a **substantial** discussion about the testing you have performed (for each test: what was to be tested, what was the expected outcome and what was the actual outcome) and which conclusions these tests allow about the correctness of your design and implementation.
- One example configuration file for the example network of Figure 1.
- Your source code.

Important:

- Please make sure that your `.pdf` file contains all fonts and can be completely printed on UC printers – if it cannot be printed it will **not be marked**. Files not submitted in `.pdf` format will **not be marked!**
- Please make sure that there is only one report submission for each pair.

The report has to be submitted via the LEARN page for COSC364. You find the submission deadline and an assignment submission box in the Assessment section on LEARN. Late submissions are **not** accepted, unless approved through the UC special consideration process.

6 Marking

Marking will be based on the following factors:

- Achieved functionality (70%). This includes: convergence to correct routing tables, robustness against station failures, syntax and consistency checks (e.g. correct packet format, correct values and ranges for numbers, handling of read accesses beyond the end of the file etc.), CPU load, ability to read filename parameter from the command line, quality and understandeability of printed output, etc. For this factor first a "raw value" is determined based purely on achieved functionality. Following this, these raw marks will be weighted against the relative contributions of each group member and individual marks will be determined.
- The quality of the report (15%).

- Your ability to explain the source code, its architecture and major functions. This is determined individually and normally weighted with 15%, but if you give the impression that you have not even been remotely present when the software has been created, you will receive significant reductions in marks or even fail the assignment, depending on the case at hand.
- When after a quick glance if the lecturer finds that your source code is particularly ugly / unreadable / un-organized, the lecturer will apply deductions of up to 10% of the achievable marks. These will be applied to both team members equally.

References

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