

COSC364

Assignment 1



April 27, 2018

Kate Chamberlin 54384616

SHAN KOO 44001993

**Percentage Contribution**

Kate 50

Shan 50

**Which aspects of your overall program (design or implementation) do you consider particularly well-done?**

The configuration parser (ConfigParser.py) and the encoding/decoding of packets. We felt that the configuration parser was well-done as it was cleanly-written code that was laid out well. It utilised code that was already in the library (configparser) to reduce complexity.  
We also felt that the way we serialised the packets was well-done (encode/decode in Packet.py). Using the struct module in the standard library was extremely useful in ensuring that the header was correctly padded. The way that we encoded/decoded our packets was also very efficient for parsing the RT entries directly into our main Router code.

**Which aspects of your overall program (design or implementation) could be improved?**

We would like to improve the functions update and update\_routing\_table. This is largely because the nested ‘for’ loops and ‘if’ statements reduced readability and made it more complex to make small adjustments to our code.  
We also thought that we could have been better with our modularisation – some of our functions contain code that could most likely have been put into more suitable functions if we had been more forward-thinking with our planning of this assignment.

**How have you ensured atomicity of event processing?**

We have ensured atomicity by running each type of event in a new thread. For example, our timers were in separate threads to our periodic updates, to ensure that none of our essential functions were blocked.

**Testing:**

Module testing:

Within the Packet.py module we tested thoroughly to ensure that the encoding/decoding worked correctly, and none of the information was lost as it was being converted in the serialization process. For this we tested empty packets, full packets, triggered-update packets and packets with invalid variables. These tests resulted in the expected outcome – a decoded dictionary containing the RTEs that we input for the tests. For example, entering RTEs {3: [4, 5], 6:[3, 2]} and encoding and decoding it resulted in {3: [4, 5], 6:[3, 2]}, and so on. We also tested the split horizon code within this module to ensure that it sent metrics of infinity to the routers it had learned the route from.  
Some values (such as the version number) we did not bother testing since the brief stated that they would be constant.

The ConfigParser.py module was tested with several input values, both valid and invalid. Similarly to the Packet.py testing, some variables that were stated as constant in the brief did not have relevant exceptions coded into ConfigParser since they were assumed to be correct. However, our testing initially showed that we had not raised our port number error correctly – port numbers below 1024 were still considered valid. We therefore altered our code and after that had no issues in the rest of our tests.

Router.py contained several more complex tests. We tested the initialisation quite thoroughly with expected results for our neighbour list, metrics, port lists, socket creation and sending empty packets to ensure that the connections were fine. For example, creating a router with a given config file would print out the neighbours, both port lists, both socket lists, and their given metrics. This showed us that there were no issues with module imports or parsing, so we then moved on to our threading.

Our thread testing involved printing an active count of our threads while code was running, and printing the output from those threads on a regular basis. Initially we were having difficulties with the threads as they kept throwing a ‘bootleg’ error that we could not trace back. This error was not fatal however and it kept looping through the code. Eventually we realised that we had made a syntactical error by putting an argument into our threaded function call, when instead we needed to parse them in as a list. Once that was working we then culled the main thread to check if our threads were still running independently and they did continue to run as expected.

Router filling/updating was the most complex as we needed to test our routing table update functions. Since they were nested it took considerably longer than our previous testing. We began by testing how the neighbours filled the routing table with expected results, and then tried to test the convergence of the entire demonstration network. The convergence was not working however, and some of the routers were maintaining routes with higher metrics than expected. This was because we were updating local variables rather than the global routing table, and once we fixed that our network converged correctly.

We then tested the robustness of our RIP implementation by culling and restarting routers. We were having difficulty at this point with our time-out working as expected, but our garbage collection being over-written with each periodic update. So when a router was dropped, the time-out would initialise, count up, and once it hit the maximum value it would set the route to infinity.