**COSC364 RIP Assignment**

Highlighted parts are all done and completed.

Team Members: Haider & Drogo

Assignment Worth: 30% Total Grade (Programming Project)

* Done under Linux Operating System (Pair Work)
* RIP Routing Protocol
* Run RIP Demon
* Project Goal: To explore responses of RIP protocol to faults
* Language Used: Python3 Programming Language (use socket interface implementation)
* Program must be executable under Linux using bash.
* Use General Discussion for any questions on LEARN.
* First Goal: To read RIP specification

**Assignment Problem Specifications**

* Implementing a “routing demon”
* Routing demon communicates with peer demons on same machine (through local sockets)
* Each “routing demon” runs individually
* Text mode program. (NO GUI REQUIRED)

**3 Stages of “Routing Daemon”**

Stage 1

* It reads a configuration file. (Name of file is supplied as a command line parameter)
* Configuration file contains: 1) Unique identification of routing demon instance 2) Port numbers on which demon receives routing packets from peer demons 3) Specification of outputs to neighboured routers
* Output port of one router is input port of another router.
* Configuration file is only informative to demons about links.
* Demons internal routing table MUST NOT be initialized from configuration file.

Stage 2

* Create same amount of UDP sockets as the number of input ports it has. (No input port of other routers to be included.)
* It binds 1 socket to each input port.
* No sockets created for output ports.
* One input socket can be used for sending UDP datagram to neighbours

Stage 3

* Enter infinite loop
* React to events in loop -> (see **select()** call or Python equivalent)
* Two types of events in loop: 1) Incoming Event – Routing packet received from peer 2) Timer Event – Send unsolicited RIP response message to peers
* For an incoming event: a) Update your own routing table b) Print on screen c) Send own routing messages to peers
* Incoming and timer event must be independent and shouldn’t interrupt the processing of each other.
* All routing demons on same Linux Machine
* Use IP for local host -> **127.0.0.1**
* Use UDP protocol for routing messages
* Design Framework: finite automata
* Will have to specify for each state what happens when an event comes in. Done for each event.

**Configuration File**

* Program should be a single executable file with single command line parameter
* This parameter is filename of configuration file
* Configuration file must be ASCII (can be read or edited)
* Syntax and consistency checks need to be made. (No need for fancy parsing)
* Each instance of router demon will be started with separate configuration file
* The file should allow to at least set the following parameters: **1) Router ID** (Unique ID of this router) (format: **router-id 1)** (**integer between 1 to 64000)** (Each router has unique ID, so the **ID parameter is different for each router’s configuration file) 2) Input Port Numbers** (Routing demon will listen for incoming routing packets from peer demons on these ports) (format**: input-ports 6110, 6201, 7345**) (separate input port for each neighbour of router) (Parser should check the following: a) All port numbers are positive integers and between 1024 – 64000, b) All entries must be in one line, c) Each port occurs only once **3) Outputs** (contact information specified for neighbour routers , directly linked) (format: **outputs 5000-1-1, 5002-5-4)** (first number specifies input port number, second value is metric value for the link to peer router, third value is router-id of peer router) (same conditions for input port numbers) (no output port numbers can be in input port numbers)

**4) Values for timers**  (these are periodic updates and timeout calls) (Use shorter timer values as less experimentation times) (ratio of timer values must remain same)

* The first three parameters (router-id, input-ports, outputs) are mandatory.
* One configuration file for each router. (Must be set up by us)

Make sure of the following conditions:

1. Router-id of each router is distinct
2. When you want two routers A and B to be neighboured, 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

**Exchanging Routing Packets and Route Computations**

* No extensions in section 4 of (RIPv2 specification) to be included.
* Implement split horizon with poisoned reverse.
* Implement triggered updates ONLY when routes become invalid.
* Use only the port numbers from the configuration file.
* Only use periodic and triggered updates. (No request messages).
* Version number is always 2.
* No need to handle IPV4 addresses or subnetworks. Only route to other routers by their router-id. No need to take care of default addresses, host routes and subnet masks.
* RIP response packet includes the entire routing table. Each neighbour is sent a copy of this table. Each neighbour gets a different view of the table (according to split horizon with poisoned reverse).
* Only send response messages through input ports of peer routers. Do not use multicast response.

**RIP Packet Format:**

* Ignore issue of network byte order
* Use 16-bit wide all-zero field in RIP packet common header for the router-id. Work with router-id’s instead of IPv4 addresses.
* Perform consistency checks on incoming packets: a) have fixed fields the right values? b) is metric in the correct range c) non-conforming packets should be dropped and a message be printed
* In python, all packets should be constructed as byte-arrays

**Periodic Updates:**

* Find out how to generate a period timer events.
* Find out how to write your own handlers for this event.
* Introduce some randomness to the periodic updates (set timer to a value uniformly distributed over the range [0.8 \* period, 1.2 \* period].

If a demon has several input ports, it needs to listen to all of them simultaneously.

Use (select() system call)

* Perform various tests with your setup
* Our tests should show that the protocol design converges to the correct routing tables.
* They respond in the right way to failure events.
* To conduct these tests, formulate in advance which behaviour our implementation should show and compare to the actual outcome against it.
* Spend time on testing and documentation as it ill play a substantial role in marking the report.