Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Spring Term 2019

Computer Networks
Project 2: Dynamic Routing with RIP

Assigned on: **6 May 2019**
Due by: **2 June 2019 at 23:59**

1 Introduction to Routing Information Protocol

The Routing Information Protocol (called RIP in the following) is a routing protocol based on the Bellman-Ford (or distance vector) algorithm. RIP has been used as the intra-domain routing algorithm in many small-scale autonomous systems (ASes) on the internet since the early days of the ARPANET. RIP is limited to networks whose longest path (the network's diameter) is 15 hops. RIP version 2 is specified in RFC 2453 but you do NOT have to implement all features supported by RIP version 2. Below is a brief description of the protocol (adapted from RFC 2453) that outlines the specific functionality that your code must support in the following. Please download RFC 2453[1] and read it for details.

1.1 RIP Basic Procedure

At a high level, the basic procedure carried out by every entity (i.e., router) that participates in the routing protocol is as follows:

- Keep a routing table (or more accurately, a forwarding table) with an entry for every possible destination network. Each routing table entry contains the destination network, the "metric" D (also called distance or "cost") to reach the destination, and the "next-hop" router G that is the next router on the chosen path to that destination.
- Periodically or due to other events as discussed in the following, send a routing update to every neighbor; these routing updates are called "RIP advertisements". The update is a set of messages that together contain all of the information from the routing table. The update contains an entry for each destination network, with the cost/metric to reach that destination.
- When a routing update arrives from a neighbor G', add the metric/cost associated with the network (i.e., link) that is shared with G'. (This should be the network over which the update arrived.) Call the resulting metric D'. Compare the resulting metric with the current routing table entries to that same destination network N. If the new metric D' for N is smaller compared to the existing metric D, then adopt the new route. That is, change the table entry for N to have metric D' and router G'. If G' is the router from which the existing route came (that is G'=G), then use the new metric even if it is larger than the old one. In case the update contains a destination N, that is not yet in the routing table, then add a new route to the current routing table with metric D' and route via G'.

[1] `https://tools.ietf.org/html/rfc2453`

More specifically, each node maintains a routing table, with each entry contains at least the following information:

- Destination address: RIP represents destination networks by a "network address" and a corresponding "network mask", both of which are IPv4 addresses. Commonly, the network address and mask together are referred to as a "network prefix" and written as X.X.X.X / Y, where X.X.X.X is the network address and Y is the length in bits of the network mask.
- Metric: Total cost of getting a datagram from the router to that destination network. This metric is the sum of the costs associated with the networks that would be traversed to get to the destination.
- Next hop: The IPv4 address of the next router along the path to the destination. If the destination is on one of the directly-connected networks, this item is not needed.
- Timestamp: Time information to determine when routing information is stale. The timestamp is the last time the router received an update for this route entry.

The entries for the directly-connected networks are set up by the router when providing the topology.

1.2 Split Horizon with Poisoned Reverse

To prevent routing loops involving two nodes from occurring and to accelerate convergence in case of link cost changes, RIP uses a scheme called "split horizon with poisoned reverse". The simple split horizon scheme omits routes learned from one neighbor in updates sent to that neighbor. Split horizon with poisoned reverse includes such routes in updates, but sets their metrics to infinity (infinity is typically set to 16, since the maximum path limit in RIP is 15). In this assignment, you should use split-horizon with poison reverse.

1.3 Triggered Updates

Split horizon with poisoned reverse will prevent any routing loops that involve only two routers. However, it is still possible to end up with patterns in which three routers are engaged in mutual deception. For example, A may believe it has a route through B, B through C, and C through A. Split horizon cannot stop such a loop. This loop will only be resolved when the metric reaches infinity and the network involved is then declared unreachable - that is, if the metric is higher than 15 remove the route entry. Triggered updates are an attempt to speed up this convergence. To get triggered updates, we simply add a rule that whenever a router changes the metric for a route in its routing table, it is required to send update (RIP advertisement) messages immediately, even if it is not yet time for one of the regular update message.

1.4 Timers

RIP has several types of timers for sending periodic updates, timing out routes, and actually removing routes from the routing table. To simplify this project, you must implemnet only the following simple timers:

- Periodic updates: Every 10 seconds, the RIP process sends an unsolicited Response message containing the complete routing table to every neighboring router.
- Route timeout: RIP maintains a TTL (time to live) field for each dynamic route (i.e., a route learned from a neighboring router. Local routes, which are directly connected networks, have an invalid TTL of -1). If a router sees an update containing a destination/next-hop pair that already exists in its local dynamic routing table, it updates the timestamp with the current time for that entry in its local routing table. If the timestamp of a route reaches is more than 20 seconds old (see "RIP_TIMEOUT_SEC" in "dr_api.c"), the route is removed from the routing table and a RIP advertisement is broadcast to all its neighbors.

2 Requirements

There are several conditions under which a router needs to send out a RIP advertisement. We summarize these below:

- Periodic Update: a router sends RIP advertisements every 10 seconds
- Triggered Update:
  - Caused by a local change in the table, i.e., an interface went down or the cost of a local interface changed. This should cause an update to the local routing table and the immediate transmission of the RIP advertisement to neighbors.
  - Caused to remote change, i.e., the router received an advertisement that caused and update to its local table. That update needs to be announced immediately.

In addition to sending RIP advertisements, your router should also handle advertisements it receives, according to the specifications in the previous section.

3 Getting Started

For the assignment you will use the LVNS server, which is located in the *code* folder. Read the included README file very carefully, as it gives specific instructions on what functions need to be implemented.

In the starter code, we have also provided five sample topologies. To create your own topology, all you need is a file that lists routers with their corresponding name and interfaces and a list of links defining how the routers are connected.

The LVNS server handles communication between the routers and will also allow you to introduce network events: link cost changes and links going down.

For this assignment, you will use a precompiled binary for the LVNS server. To ensure a working environment, we suggest to use the same VM as in the last project. Link to the virtual machine: `https://ndal.ethz.ch/external/blank-ubuntu-networks.zip`.

To make it easier for you to run dynamic routing (dr) instances, you can use the launch_dr.sh script to connect to your lvns server. It pipes the output from your dr instances into log files.

```
ethz1:~> ./lvns -t complex.topo
# In another terminal on the same server
ethz1:~> ./launch_dr.sh 5
```

All this is explained in detail in the README file accompanying the code.

4 Testing the assignment

We do not provide you with testing scripts. While implementing this assignment, you should use pen and paper to walk through your topologies and understand how routing tables change as the state of the network changes.

However we did add some helper functions so that it's easier for you to debug your code.

We will grade this project by running your code with our own topologies and checking your resulting routing tables.

There may be a small penalty for wrongly submitted code (e.g., wrong folder structure).

5 Acknowledgements

This Project has been adapted from Stanford's CS144 Introduction to Computer Networking Labs.

6 Plagiarism

Note that we will check your submissions for plagiarism.

7 F.A.Q.

Do we need to implement the garbage routes?
Optional. You may delete entries upon timeout.

What is the next hop of a directly connected subnet?
0.0.0.0

What is the destination IP address and next hop for a RIP advertisement?
Use the defined RIP_IP constant for the destination IP address and next hop.

What next_hop_t should I return if there is no existing route?
You should return 0xFFFFFFFF for `dst_ip` in this case since 0 is reserved for a directly connected network. The value of the `interface` field does not matter here.

What byte ordering is used?
In general, any IP addresses or subnet masks passed to or returned by any of the `safe_dr_*` functions are in network byte order. Any other argument or return value is in host byte order. Note that `print_ip` expects an IP address in host byte order.

It's up to you how you store things internally in your routing table. You may need to adapt how the `print_routing_table` prints IPs if you store them in network byte order.

Will we have to implement route aggregation?
While you have to perform longest prefix matching, doing route aggregation is not necessary.

Can a router interface be connected to multiple interfaces?
No. In this project you can assume that point-to-point links are used; an interface is connected to exactly one interface.

Is it guaranteed that the interface given in `safe_dr_handle_packet` is enabled?
No, an interface that is down may still receive routing packets. You need to explicitly check this and ignore such routing packets.