Assignment 3 - Routing Protocols

1 Goal

The goal of this project is for you to learn to implement distributed routing algorithms, where all routers run an algorithm that allows them to transport packets to their destination, but no central authority determines the forwarding paths. You will implement code to run at a router, and we will provide a routing simulator that builds a graph connecting your routers to each other and simulated hosts on the network. To get started, you will implement a learning switch, which learns the location of hosts by monitoring traffic. At first, the switch simply broadcasts any packet it receives to all of its neighbors. For each packet it sees, it remembers for the sender S, the port that the packet came in on. Later, if it receives packets destined to S, it only forwards the packet out on the port that packets from S previously came in on. Recall from class that a learning switch is not a very effective routing technique: it breaks when the network has loops, cannot adapt to routing failures, and is not guaranteed to find efficient paths. Hence you will then implement a RIP2 -like distance vector protocol to provide stable, efficient, loop-free paths across the network. Routers share the paths they have with their neighbors, who use this information to construct their own forwarding tables.

2 Simulation Environment

Figure 1 and Figure 2 describe only the classes you will need to implement. Your LearningSwitch and RIPRouter will extend the Entity class. Each Entity has a number of ports, each of which may be connected to another neighbor Entity. Entities send and receive Packets to and from their neighbors. The Entity superclass has four functions that will be relevant to you (there are a number more that you can feel free to peek at, but these should be sufficient to complete the assignment):

```
class Entity(__builtin__.object)
  handle_rx(self, packet, port)
    Called by the framework when the Entity self receives a packet.
    packet - a Packet (or subclass).
    port - port number it arrived on.
    You definitely want to override this function.

  send(packet, port=None, flood=False)
    Sends the packet out of a specific port or ports. If the packet's
    src is None, it will be set automatically to the Entity self.
    packet - a Packet (or subclass).
    port - a numeric port number, or a list of port numbers.
    flood - If True, the meaning of port is reversed - packets will
    be sent from all ports EXCEPT those listed. Do not override this function.

  get_port_count()
    Returns the number of ports this Entity has.
    Do not override this function.

  set_debug(*args)
    Turns all arguments into a debug message for this Entity.
    args - Arguments for the debug message.
    Do not override this function.
```

Figure 1: Entity Class components

```
class Packet(object)
  self.src -> The origin of the packet.

  self.dst -> The destination of the packet.

  self.ttl -> The time to live value of the packet.
  Automatically decremented for each Entity it goes.

  self.trace -> A list of every Entity that has handled the packet previously.
  This is here to help you debug.
```

Figure 2: Packet Class components

3 Learning Switch Specification

You will write a LearningSwitch class which inherits from the Entity class, overriding the handle_rx function. If a packet comes in destined to an Entity from whom you have never seen a packet before, broadcast the packet on all ports except the port which the packet came in on. If a packet arrives destined to an Entity from whom you have previously received a packet, only send the packet out on the port from which packets from that Entity previously arrived. Your LearningSwitch does not need to deal with links coming up or down, or networks with loops.

4 RIP Router Specification

Write a RIPRouter class which inherits from the Entity class. The function that you will need to override is handle_rx dealing with the following three types of packets.

- DiscoveryPackets are received by either ends of a link when either the link goes up or the link goes down.
- RoutingUpdates contain the routing information that is received from the neighbours.
- Other packets are data packets which need to be sent on an appropriate port based on switch's current routing table.

DiscoveryPackets are sent automatically to both the ends of a link whenever the link goes up/down (the boolean variable is_link_up signifies whether the link has gone up or has gone down). You must handle these packets properly, i.e. receiving a DiscoveryPackets with a link up event signifies a 0-hop path (direct link) to the source of the packet. Your RIP Router should maintain a forwarding table and announce its paths to its neighbors using the RoutingUpdate class (defined in sim/basics.py and extends Packet) and contains a field called paths which is a hash-table mapping advertised destinations to the distance metric. Your RIP Router implementation must use the RoutingUpdate class as specified to share forwarding tables between routers (otherwise it will not be compatible with our evaluation scripts and you will lose points, **even if your RIPRouters are compatible with each other.** Your implementation should perform the following:

- Routing preferences. On receiving RoutingUpdate messages from its neighbors, your router should prefer (1) routes with the lowest hop count, (2) for multiple routes with the same hop count, it should prefer routes to the neighbor with the lower port ID number.
- Dealing with failures and new links. Your solution should quickly and efficiently generate new, optimal routes when links fail or come up.
- Implicit withdrawals. RoutingUpdate packets should contain a list of all paths a router is willing to export. If a router previously announced a path, but a later update does not contain the announced path, the path is implicitly withdrawn.
- Split Horizon with Poison Reverse. To prevent routing loops, your router should announce 'poison reverse' announcements with hop count values of 100.

We will test your learning switch to verify that it broadcasts and learns as specified. We will evaluate your RIP router on correctness, number of routing messages propagated, and how long it takes for new paths to stabilize in case of link failures (convergence time). Your RIP routing protocol must converge on shortest paths and must not deliver packets to hosts other than the one they were destined to.

5 Extra Credits

An alternative to Distance Vector algorithms is a Link State design. Instead of exchanging best paths, nodes exchange lists of all of their neighbors so that each node in the network has a map of the entire network topology. Each node then independently calculates the best path across the network for each destination. Implement a router that uses a Link State algorithm called LSRouter. Benchmark the convergence time (time between a link failure/link coming up and all of the nodes coming to a stable set of new paths) of your LSRouter against the convergence time of your RIPRouter.
Turn in: Your implementation and an analysis of the convergence times: which performs better, and why?

6 Submission

- To be done in the group of two.
- You must also supply a README.pdf file along with your solution. This should contain:
  - What challenges did you face in implementing your router?
  - If you implemented extra credit question; describe what they do and how they work.
- Create a tar file contatining learning_switch.py, rip_router.py and README.pdf.
- Rename the tar file as Your_Entry_Number.*.
- Submission deadline is Sep 22 11.55 PM.