Routing Simulator Guide

1 File Layout

The simulator is organized thusly:

- simulator.py - Starts up the simulator.
- sim/core.py - Inner workings of the simulator. Keep out.
- sim/api.py - Parts of the simulator that you'll need to use (such as the Entity class). See help(api).
- sim/basics.py - Basic simulator pieces built with the API. See help(basics).
- sim/topo.py - You can use this to create your own topologies/scenarios. See help(topo).
- scenarios/*.py - Test topologies and scenarios for you to use. See help(scenarios).

2 Simulation Environment

To start the simulator, run simulator.py to interact with the simulated network:

```
$ python simulator.py
```

This spouts some informational text, and gives you a Python interpreter in the terminal. From the interpreter in the terminal, you can run arbitrary Python code, can inspect and manipulate the simulation, and can get help on many aspects of the simulator. As simulator.py comes, it simply creates a test scenario (mostly this just means a topology) and starts the simulator. You may wish to modify it to load a different scenario (including a custom one). In particular, you can modify the scenario on line 22, and the switches used in the network on line 15. By default, the simulator will load a 'linear' scenario (two hosts connected by a single hub) with a Hub switch that just forwards between the two. To change the scenario and switches imported by the simulator, find the lines that define 'switch' and 'scenario' and edit them, e.g.,

```
from learning_switch import LearningSwitch as switch
from rip_router import RIPRouter as switch
import scenarios.linear as scenario import scenarios.candy as scenario
```

You can see the source code for the Hub switch in hub.py, and the source code for the scenario in the scenarios/ directory.

The scenarios in the scenario directory each contain a create() method. You might want to look at these. They create the topology that goes with the scenario. The included ones can be run with an arbitrary switch class, so you can easily set them up using a hub, a leaning switch, your RIP-like switch, etc.

From the simulator's commandline, you have access to all the Entities created in your scenario, and you can interact with these. For example:

```
>>>start() → make sure you don't forget this.
>>>h1.ping(h2)
```

To get started, we suggest you play around with the default scenario and default switches and make sure you understand how to send pings between hosts, add DEBUG messages to the log, and understand the DEBUG messages printed by default to the log viewer.

3 The Log Viewer

Log messages are generally sent to the terminal from which the simulator is run. If you are using the interactive interpreter for debugging or experimentation, this can be somewhat irritating. You can disable these by editing simulator.py towards the top, there is a line: #_DISABLE_CONSOLE_LOG = True.

If you uncomment this line (by removing the '#') you will no longer see log messages on the terminal. Of course, those log messages can be really quite helpful, and you might want to see them. To rectify this, there is a standalone log viewer logviewer.py. Run this along with the simulator as:

```
$ python logviewer.py & python simulator.py
```

Executing this program should opens a new window where you'll see DEBUG messages output by the hosts and switches in the simulated network.

4 Implementing Entities

Objects that exist in the simulator are subclasses of the Entity superclass (in api.py). These have a handful of utility functions, as well as some empty functions that are called when various events occur. You probably want to handle at least some of these events! For more help try help(api.Entity) within the simulator. You might also want to peek at hub.py, which is a simple Entity to get you started.

4.1 Sending Packets

Entities can send and receive packets. In the simulator, this means a subclass of Packet. See basics.Ping for one such example, and see basics.BasicHost to see an example of how it is used. In the default simulation (a linear topology with a hub), try h1.ping(h2) to start.

5 Building your own schenarios

You may want to test using your own topologies, and you may want to test your own events (such as nodes joining and leaving the network) within those topologies. We refer to the combination of those as a 'codescenario'. The simulator should come with some (in the scenarios directory) to get you started, but you may want to build your own. To start:

```
>>>import sim.topo as topo
```

The first step is simply creating some Entities so that the simulator can use them. You shouldn't create the nodes yourself, but rather let the create() function in core do it for you, something along the lines of:

```
>>>MyNodeType.create('myNewNode', MyNodeType, arg1, arg2)
```

That is, you don't want to use normal Python object creation like:

```
>>>x = MyNodeType(arg1, arg2) # Don't do this
```

create() returns the new entity, and all entities that you create are added to the sim package. So you can do:

```
>>>import sim
>>>x = MyNodeType.create('myNewNode', arg1, arg2)
>>>print sim.myNewNode, x
```

which will show the new Entity twice.

To link this to some other Entity:

```
>>>topo.link(sim.myNewNode, sim.someOtherNode)
```

You can also unlink it, or disconnect it from everything:

```
>>>topo.disconnect(sim.myNewNode)
```

To see the connections on a given Entity:

```
>>>topo.show_ports(sim.someNode)
```

this shows how the ports on some node are connected to other nodes and their ports.