Assignment 7: Best Goodput

Due date: *Sunday, May 31, 2025 at 22:00*

> *This is an individual assignment. You may discuss it with others, but your code and documentation must be written on your own. In particular, do not use automated code-generation tools such as ChatGPT or GitHub copilot.*

In a source file called `best_goodput.py` write a Python program that takes as input a YAML file containing a network topology definition and some flow demands, and creates a Mininet emulation with the network configured to obtain the best overall goodput under the given set of demands. The general usage of the program should be the following

```
$ ./best_goodput.py --help
usage: best_goodput.py [-h] [-p] [-l] definition

A tool to define the emulation of a network configured to achieve the best overall
goodput under a given set of flow demands.

positional arguments:
  definition   the definition file of the network and flow demands in YAML

options:
  -h, --help   show this help message and exit
  -p, --print  print the optimal goodput for each flow and exit
  -l, --lp     print the definition of the optimization problem in CPLEX LP format
```

As in the previous assignment, the topology definition consists of two sections, *routers* and *hosts,* as shown in the example of Figure 1. The routers section (`routers:`) specifies a number of routers, each identified by a unique name (e.g., `r1` in Figure 1). You may assume that no router name matches the regular expression `s[0-9]+` (explained later). A router has zero or more interfaces. Each interface is characterized by an IPv4 address, a network mask, and an optional cost associated with the link. Each link cost is measured in Mbps. The cost of a shared link can be specified with any of the interfaces that share that link. If no cost is specified, the default should be 1Mbps. You may assume that the cost specifications are consistent (same cost for adjacent interfaces).

The hosts section (`hosts:`) specifies a number of hosts, each identified by a unique host name that also does not match the regular expression `s[0-9]+`. A host has one interface, characterized by an IPv4 address and a network mask. The host interface is guaranteed to be in a subnet with a single router interface, which you can therefore use as default gateway for that host.

You are guaranteed that the input topology definition file is valid. For instance, the router and host sections will always be present, the file will be in valid YAML format, there will be no duplicate IP addresses, etc.

The topology definition is such that every interface is connected to at least another interface from some other node (router or host). If multiple hosts share the same subnet address, you must add a layer-2 switch to interconnect them. The switches will not be listed in the configuration file. You can name those switches `s1`, `s2`, etc. This is why the definition guarantees that hosts and router names do not conflict with such switch names. You are guaranteed that all routers will be interconnected using point-to-point links.

Finally, you will have a demands section (`demands:`) specifying a number of flow demands. You can think of a flow demand as a triple (s, d, r) where s is the source host, t is the target host, and r is the

```yaml
---
routers:
  r1:
    eth0: {address: 10.0.12.1, mask: 255.255.255.252, cost: 3}
    eth1: {address: 10.0.13.1, mask: 255.255.255.252, cost: 10}
    eth2: {address: 10.0.1.1, mask: 255.255.255.0}
  r2:
    eth0: {address: 10.0.12.2, mask: 255.255.255.252}
    eth1: {address: 10.0.23.1, mask: 255.255.255.252, cost: 8}
    eth2: {address: 10.0.24.1, mask: 255.255.255.252, cost: 10}
  r3:
    eth0: {address: 10.0.13.2, mask: 255.255.255.252}
    eth1: {address: 10.0.23.2, mask: 255.255.255.252}
    eth2: {address: 10.0.34.1, mask: 255.255.255.252, cost: 10}
    eth3: {address: 10.0.3.1, mask: 255.255.255.0}
  r4:
    eth0: {address: 10.0.24.2, mask: 255.255.255.252}
    eth1: {address: 10.0.34.2, mask: 255.255.255.252}
    eth2: {address: 10.0.4.1, mask: 255.255.255.0}

hosts:
  h1:
    eth0: {address: 10.0.1.2, mask: 255.255.255.0}
  h2:
    eth0: {address: 10.0.1.3, mask: 255.255.255.0}
  h3:
    eth0: {address: 10.0.3.2, mask: 255.255.255.0}
  h4:
    eth0: {address: 10.0.4.2, mask: 255.255.255.0}

demands:
  - {src: h1, dst: h4, rate: 10}
  - {src: h4, dst: h2, rate: 2}
  - {src: h3, dst: h4, rate: 15}
```

Figure 1: Example topology definition in YAML format.

overall transmission rate. A flow demand is specified as an item in the `demands:` list. For each demand you will have the source, the destination, and the transmission rate at the `src:`, `dst:`, and `rate:` keys respectively. You are guaranteed that the source and the destination of each flow demand is contained in the `hosts:` section.

To parse the input YAML file, you may use the *PyYAML* python package. The `requirements.txt` file available on the iCorsi system lists all the external packages that you are allowed to use[1]. You may not use any other package or non-standard library function. To install these Python packages into a separate environment you might want to create a virtual environment[2].

The goal is to optimize the overall goodput of the network under the given flow demands. The goodput is the effective data transmission rate as seen by the receiver for each flow. So, for a flow demand (s_i, d_i, r_i), r_i is the maximal rate at which the sender (s_i) transmits data. Let $r_i^* \leq r_i$ be the goodput, that is the long-term average rate at which the receiver (d_i) receives data for flow i. On this basis, we define a statistic that represents the overall network throughput. A meaningful statistic is the minimal effectiveness ratio:

$$\min_i \frac{r_i^*}{r_i}$$

The goal, then, is to maximize the minimal effectiveness ratio.

[1] You may want to use the `pip install -r requirements.txt` command to install all the packages you can use with their versions.

[2] https://docs.python.org/3/library/venv.html

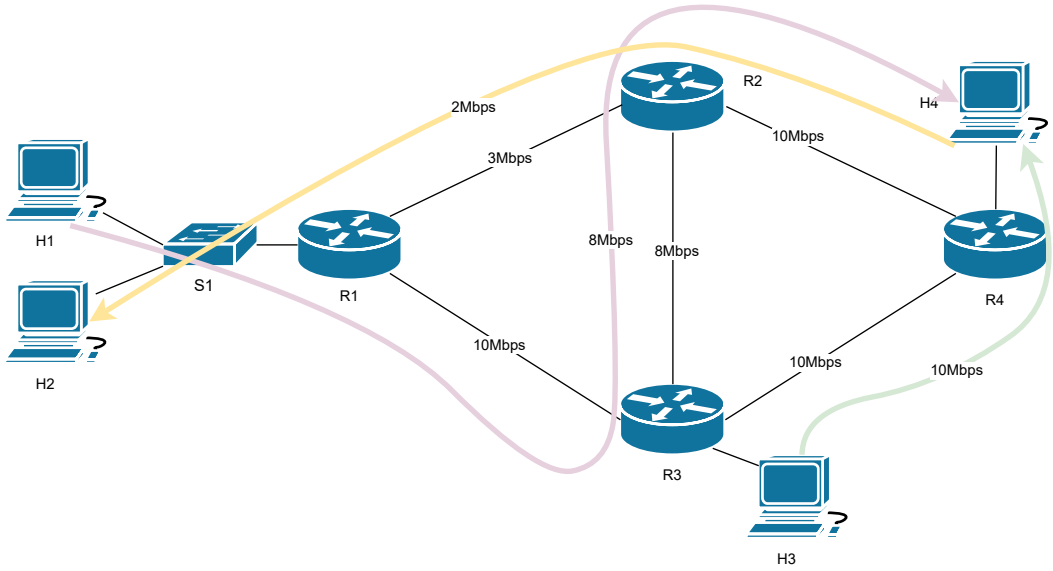

Figure 2: Visualization of the optimized routing scheme.

Another way to look at this goal is to consider each demand in terms of data size for a fixed target time, rather than rate. In a target time interval T, each demand (s_i, d_i, r_i) requests the transmission of $F_i = r_i T$ bytes of data from s_i to d_i. Assuming that all transmissions start at the same time, all transmissions of F_i bytes at rate r_i should terminate after time T. Your goal is to minimize the overall (maximal) completion time.

For example, Figure 2 illustrates the network given by the definition file in Figure 1, together with the routing scheme that maximize the minimal effectiveness ratio. It can be seen that $r_1 = 0.8$, $r_2 = 1$ and $r_3 = 0.66$, therefore

$$\min_i \frac{r_i^*}{r_i} = r_3 = 0.66$$

When called with the `--lp` option, the program must write onto the standard output the definition of optimization problem in CPLEX LP format. For instance, the program should print something similar to the following:

```
$ ./best_goodput.py --lp topology.yaml
Maximize
 obj: ...
Subject to
 ...
End
```

When called with the `--print` option, the program must list the optimal goodput achievable for each of the flows listed into the `demands:` section of the input definition file. For example, using the definition file in the example of Figure 1, the program should print something similar to the following:

```
$ ./best_goodput.py -p topology.yaml
The best goodput for flow demand #1 is 8 Mbps
The best goodput for flow demand #2 is 2 Mbps
The best goodput for flow demand #3 is 10 Mbps
```

With no parameters, the program should start a Mininet emulation. The emulation must be set up so as to obtain the best overall goodput under the provided flow demands. In practice, this means that you must properly set up MPLS rules on every router to achieve the optimal goodput.

To solve the optimization problem, you must use GLPK. That means that you have to run the `glpsol` utility, and parse its output to determine the results of the optimization problem.

Submission Instructions

Submit a source file named `best_goodput.py` through the iCorsi system. Do not submit other files. Add comments to your code to explain sections of the code that might not be clear. You must also add comments at the beginning of the source file to properly acknowledge any and all external sources of information you may have used, including code, suggestions, and comments from other students. If your implementation has limitations and errors you are aware of (and were unable to fix), then list those as well in the initial comments.

You may use an integrated development environment (IDE) of your choice. However, *do not submit any IDE-specific file*, such as project description files.