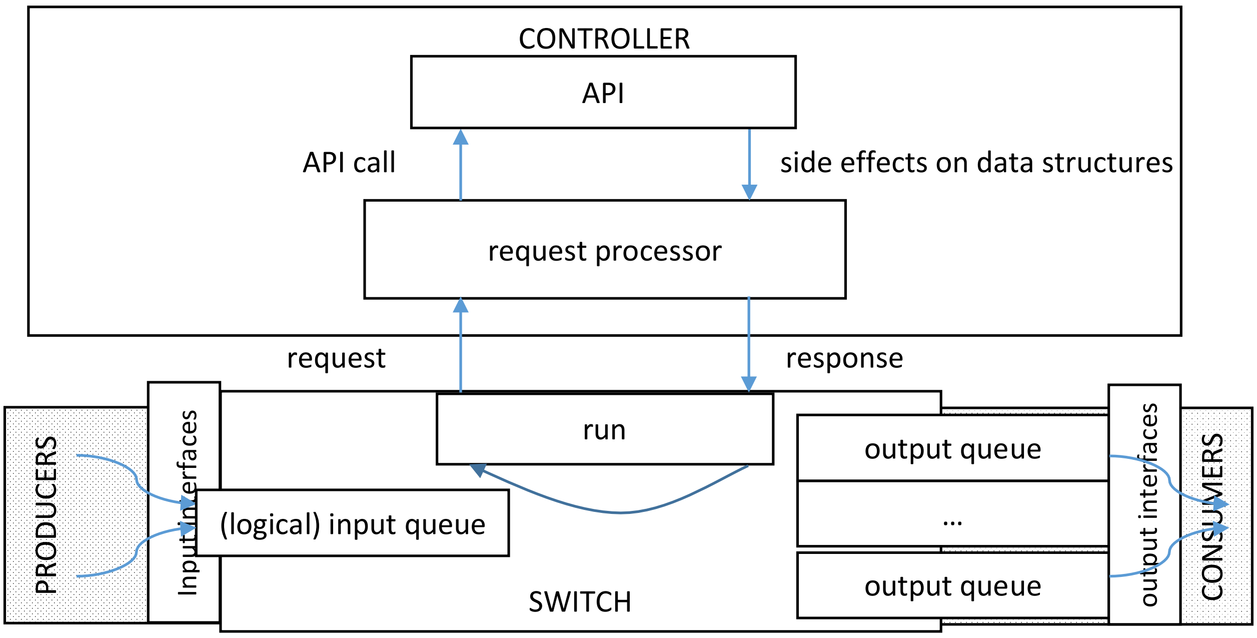
Advanced Programming, a.y. 2016/2017
Homework 2
Student: Orlando Leombruni, *matricola* 475727

Quick start

The project implements a (simulation of a) software-defined network in which the controller unit (controller.py) provides its services through the use of a Python API (lib.py). The network is simulated through a number of abstract producer (producer.py) and consumer (consumer.py) entities; every producer generates a packet and pushes it into the input interface of the switch (switch.py), which then forwards it onto the correct output interface (according to rules set by the controller). Every consumer is attached to an output interface and consumes (stores) every packet that has been put in that interface.
The simulation can be executed by invoking the Python 3 interpreter on the simulation.py file.

The Network Modelization



Please note that to show the asynchronous nature of the project, every producer and every consumer is a separate, autonomous thread.

The Simulation

The simulation is contained in the simulation.py file. It creates two producers and five consumers; the second producer starts its operations one second after the first one, and both produce a packet every two seconds. The packets are forwarded by the switch from its input queue to the correct output queue according to the routing table provided by the controller; every two processed packets, the switch requests an update to the controller. After 21 requested updates, the simulation ends. The delay between the two producers and their service time can be adjusted using the "--time ser_time delay" optional command-line argument.
The simulation can also run in a step-by-step mode, in which it stops after every producer cycle and waits for an input by the user. The user can either advance the computation, terminate it immediately, resume the automatic mode, or check the state of the network. The step-by-step mode is activated by the "--step" command-line argument.

Optional Assignment

Security Policies

Security policies in this implementation can be of three types: limit the number of “read”, “write” or both operations between a source and a destination. Policies are global, i.e. applied to every src/dest pair, and maintained through a Python dictionary that uses those pairs as keys (and (remaining_reads, remaining_writes) pairs as values).
At the beginning, the global policy is put into the dictionary's special POLICIES key; it is used as a "blueprint". Every time a packet is processed, its (source, destination) pair is checked; if this is the first encountered packet with such a pair, the global policy is then copied into the dictionary using (source, destination) as a key, then the policy is enforced and the dictionary is accordingly updated. If, instead, the pair is already a valid key (i.e. a packet with the same pair was already processed) then the previous state is read from the dictionary and the policy enforced.

Filters

Filters are implemented as Python functions from packet to booleans; this gives the programmer maximum freedom of choice. If a packet satisfies the conditions of a filter, it is dropped.

Mobility

Mobility is handled through a dictionary that has “old” IP addresses as keys and “new” IP addresses as values. Every time a packet is processed, if its destination is a valid key for the “redirects” dictionary, it is forwarded to the output port assigned to the corresponding value.

Concluding remarks

The project is written in Python 3 (specifically, version 3.5.2), using the PyCharm IDE. Python 3 itself has been installed through Homebrew on macOS Sierra 10.12.2. While I wrote the entirety of my code, the solution has been designed and debated with the help of fellow students Giuseppe Astuti, Alessandra Fais and Davide Scano.