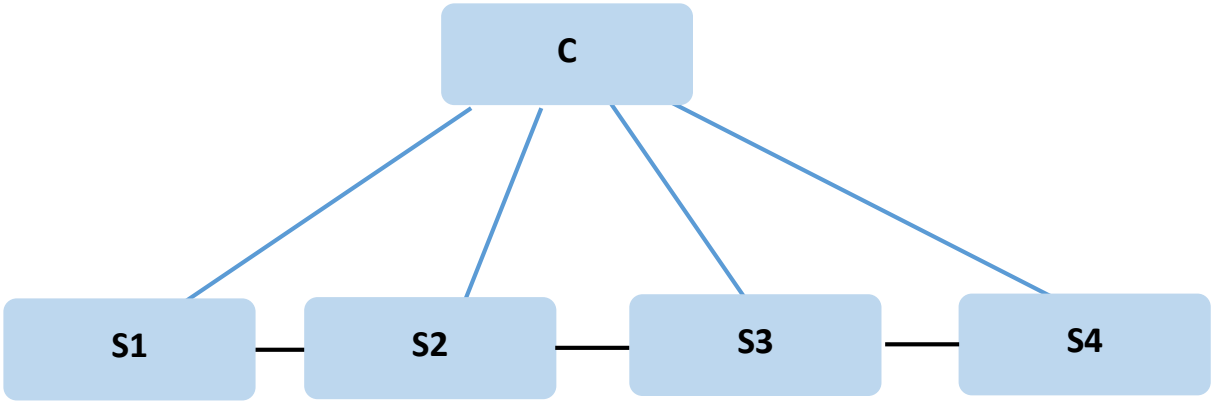
Exercise 6 – A Python SDN Simulator

1. A basic Software-defined Network simulator (100P)

(100P) *Note that this exercise will be affecting your final grade.* Now that you have recapped the basics of Python, let's get back to the course content! To wrap up the first part of the block course, in this exercise, you will build your very own (VERY basic) SDN simulator. Suppose you have the following topology:



We call this topology, in which the four switches (S1-S4) are connected in a single line, a **linear** topology. The switches are orchestrated by a single controller C. Your task is to rebuild this topology and simulate the exchange of packets over that topology in a Python simulator. Consider the lifetime of the very first packet in the network that will be send from, for instance, S1 to S4:

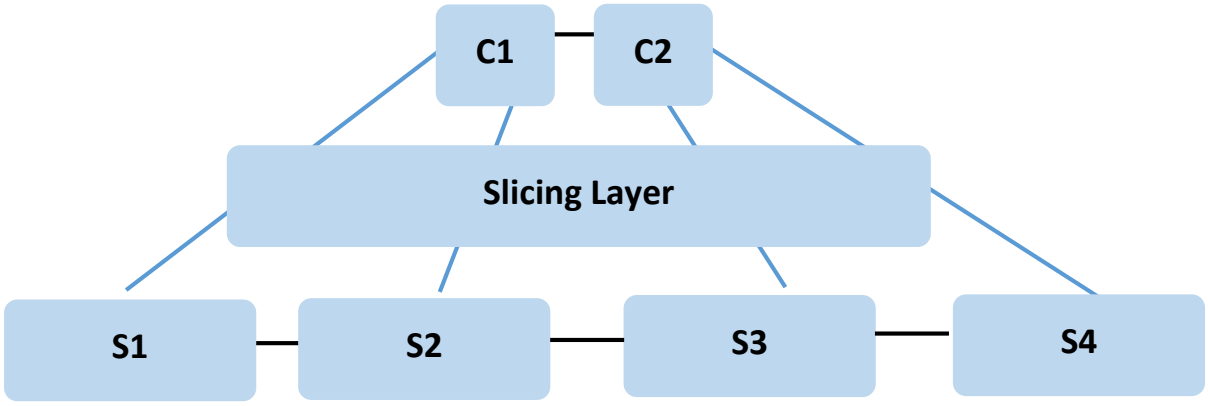
The packet will be created at S1, and since it is the first packet, S1 will not have any flow tables installed. Instead, it will have to ask the controller C for a new rule. C has to install a new rule in S1. The rule should be: "Forward to S2". S2 has to ask C for a new rule as well. Note that in this case, the rule should be "If received from S1, forward to S3". Ultimately, the packet will arrive at S4, which should know that it is the destination of the packet, and acknowledge the reception of that packet.

After you have implemented the simulator, test your system with two flows:

- one flow of five packets originating at S1 and with S4 as destination
- a second flow consisting of three packets from S4 to S1

2. Extend your simulator (50P)

Note that this exercise will be affecting your final grade. Now that you have implemented the basic simulator, extend it so that it offers some basic slicing capabilities similar to FlowVisor. In particular, you want to have an intermediate layer between your switches and controllers that checks, for each flow rule that a controller sets up, if it is allowed to do so, and for each event that is raised by a switch, forwards it to the appropriate controller instance.



Here, a slice definition should simply declare the network elements that this slice has access to. So, for instance, an exemplary slice A in the figure above could be defined as A = (S1, S2, S3), and this slice should only be able to access the flow tables of switches S1, S2, and S3. Also, you will need to make sure that your network slices are disjoint, since no slices are allowed to work on the same set of traffic (in this simplified case: same switches, since we have no finer grained control).

Test your implementation by instantiating two controllers C1 and C2, where C1 will control S1 and S2, and C2 will control S3 and S4 (as shown in the figure above). Make sure that if C1 wants to install a flow rule on S1 and S2, it will succeed, and errors are thrown if it tries to access S3 or S4.

Also, test your system again with sending flows from S1 to S4 and vice versa, and make sure that the events raised by switches S1 and S2 are dealt with by controller C1, and the events raised by switch S3 and S4 are dealt with by controller C2.

3. Hints for the SDN Simulator (Task 1) (0P)

1. The entire simulator can be written in around 100 lines of code
2. You do NOT need to build a graphical interface. Rather, build a text simulator.
3. You do not need to import any additional python modules
4. The basic classes you will need are (in one exemplary implementation):

    - **Switch** – A switch will process a packet according to its `flow_table`
    - **Controller** – The controller will setup the switches' `flow_table`s
    - **Packet** – the object that is passed between switches and the controller

5. Instantiate these objects for the elements of your topology and use methods and attributes to connect them with each other.
6. You can make use of the fact that each switch only has two neighbours at maximum.
7. The controller needs to decide the forwarding action for each switch. Each switch has a `flow_table` that should be modified by the controller. In your simulator, the modification can be done by the switch, but the controller has to make the decision.
8. At the beginning, the flow tables are empty at each switch.
9. Your code should also work if we extend the linear topology by any number of switches.