INFO8011: Network Infrastructures Software-Defined Networking in Data Centers

Ben Mariem Sami, Soldani Cyril, Mathy Laurent

1 Context

In this assignment, you will extend your learning switch obtained during the 3rd lab on OpenFlow to implement the control logic of a basic data center topology.

1.1 Data Center Topology

As discussed in the first two labs, a conventional topology in data-center networks is the hierarchical tree. In this architecture, a number of compute nodes are co-located inside a rack, and connected though a top-of-rack-switch. Then, this switch has one uplink connection towards an aggregation switch, connecting several racks. Finally, the aggregation switches themselves have uplink connections to core switches. However, efficient communication in such an architecture can only be guaranteed if the capacity of the links in the topology increase as we go higher in the tree. Indeed, the traffic that flows through the topology is both horizontal, with the advent of map-reduce style jobs inducing inter-rack communication, and vertical, with communications to the outside networks. Thus, the topology should enable full-bandwidth communications both horizontally and vertically.

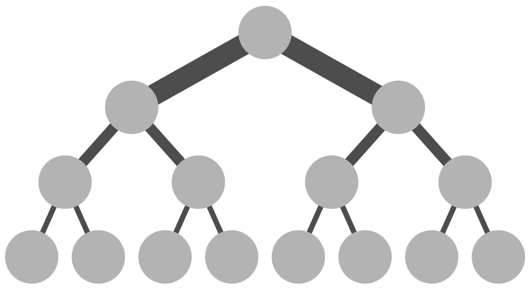

Figure 1 – Hierarchical Fat-Tree topology

Figure 1 represents an architecture attempting to provide a high bandwidth access to servers in the data center. Such a topology is commonly referred to as fat-tree[1]. Unfortunately, this topology rely on high-capacity switches and links in the upper layers of the tree to handle the increasing number of flows as you go up in the tree. However, the complexity in the engineering of such high-capacity switches and links makes them very expensive. This cause problem for stock management and maintenance as you are not able

[1]https://en.wikipedia.org/wiki/Fat_tree

to afford keeping some in stock to replace defective ones. In addition to that, the switches needs to be duplicated for redundancy reasons so that the data-center connectivity can be maintained in case of failure. However, those additional switches are generally not used, leading to a waste of expensive resources.

1.2 Clos Network

The clos network[2] topology allows to mitigate those problems by a enabling a high-capacity architecture with the use of identical switches. A clos network is a multistage switching network, first formalized by Charles Clos in 1952 in the context of circuit switching for phone communications. In clos networks, the number of switches that will be used for handling a same number of hosts is higher than for the architecture described in Section 1.1. However, identical, potentially cheap and readily available, switches can be used for an overall decreased cost with higher resiliency.

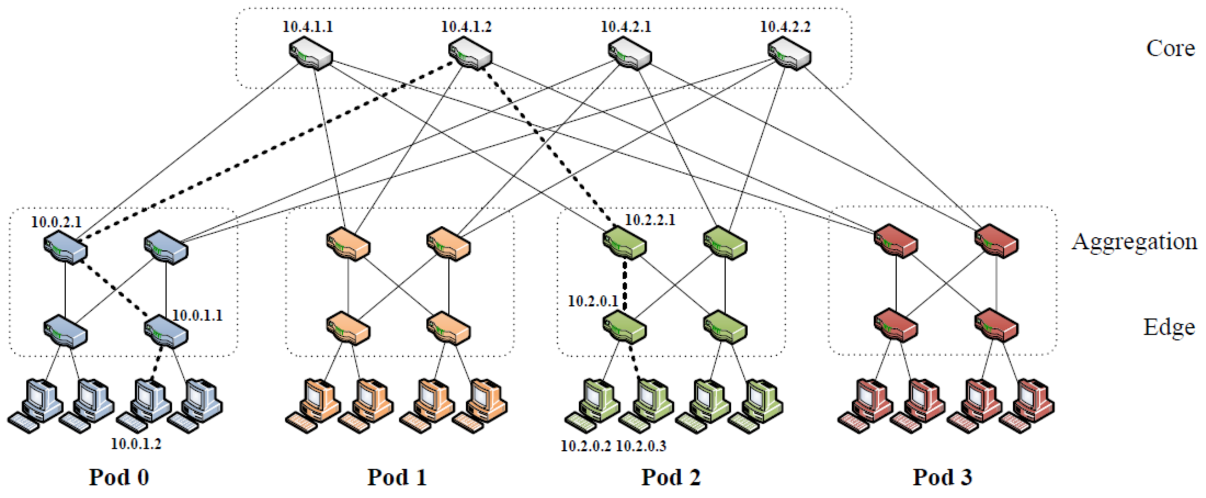

Figure 2 – Al-Fares et al Fat-Tree topology.

Figure 2 represents a topology proposed by Al-Fares et al [1], also wrongly called fat-tree (it contains cycles), which adapt the Clos network topology to packet-based data centers. This topology has the advantages that it has identical bandwidth at any bisection with each layer having the same aggregated bandwidth. In addition to that, it brings very high scalability as using k-port switches allows the topology to support $k^3/4$ servers.

Al-Fares et al Fat-tree is structured into pods which allows to horizontally scale the number of hosts handled by the network without having to use switches with higher port densities. All in all, an operator has two choices to scale its network: **(1) Vertically** by using switches with higher port densities, **(2) Horizontally** by adding pods considering the limit imposed by the port density of the switches used. For instance, 4-port switches only allows to scale up to 4 pods. Please read the paper for a more detailed description of the architecture.

[2]https://en.wikipedia.org/wiki/Clos_network

2 Assignment

Your task is to implement the control logic of clos-like networks using **mininet** and the **Ryu** framework. In this context, you will implement three control policies each enabling your SDN controller to engineer the traffic that flows through your network.

2.1 Topology

The topology that you will need to handle is provided in file FatTreeTopo.py which contains an implementation of the Al-Fares topology. Thus, your Ryu controller can be tested using a similar procedure than in the 3rd lab:

1. Run your Ryu controller in one terminal:

```
student@info8011-vm-t1:~$ ryu-manager {controller}.py
```

2. Run the Mininet topology in another terminal:

```
student@info8011-vm-t2:~$ sudo python3 FatTreeTopo.py --k 4 \
                            --nbPods 4 --bw 100 --delay 50 --maxQ 20
```

The topology that will be generated is illustrated in Figure 3

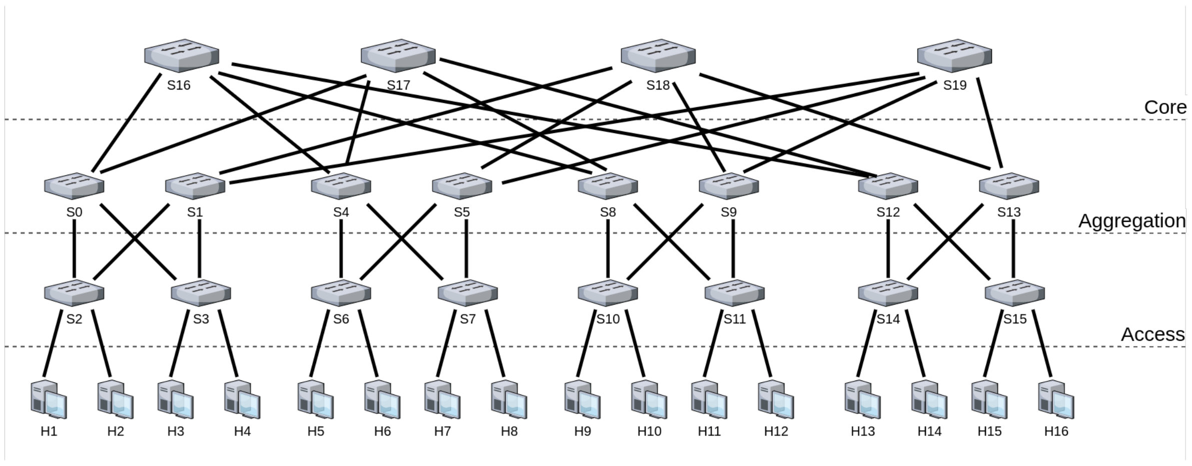

Figure 3 – Clos-like topology with 4 core switches and 4 pods

2.2 Control policies

In this assignment, you will need to experiment and study **three** different control policies for handling the Al-fares topology. One of the main challenge in handling such topologies, is to handle broadcast storm that may arise due to the presence of switching loops in the network. In this context, you are asked to provide Layer 2 switching algorithms to enable any peer in the network to efficiently communicate with any other peer. Ideally, your SDN controller should be able to discover the topology and react to live topology changes -*e.g.,* switch or links that fails. Please have a look at this Section of the documentation for relevant events callback. For each of those control policies, you are expected to test, assess and discuss the design and your implementation using all the measurements that seems relevant.

2.2.1 Spanning Tree Controller

The first policy that you will need to implement enables your SDN controller to build a **spanning tree** to handle cycles in your topology. You are not asked to implement the actual **Spanning Tree Protocol (STP)** but a a similar logic in your controller by avoiding the use of some links and switches as illustrated in Figure 4

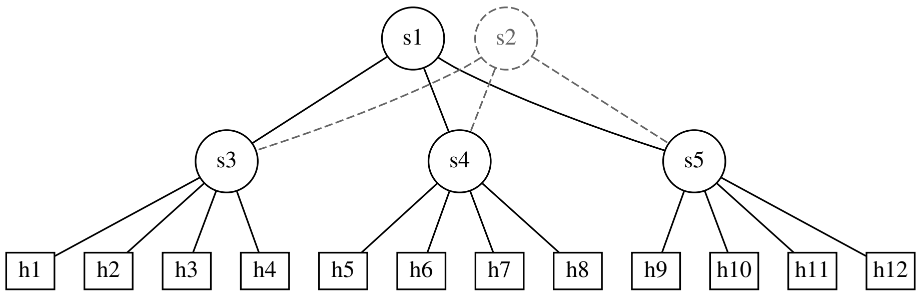

Figure 4 – The spanning-tree control policy ignores some links and switches.

2.2.2 Using VLANs

The second policy that you will need to implement enables your SDN controller to create **VLANs**. VLANs are designed to partition a single Layer 2 network in a way that creates two or more distinct broadcast domains. This allows traffic to remain isolated, even if the flows share physical hardware. Indeed, each host will belong to a single tenant, and he will only be allowed to communicate with hosts belonging to that tenant. Hosts that belong to similar tenant are grouped into VLANs and spanning trees can be used to handle the remaining cycles.

In the example illustrated in Figure 5, the network has been divided into 4 VLANs: **(1)** the red and green that are rooted at switch **s1** and, **(2)** the cyan and magenta VLANs that are rooted at **s2**.

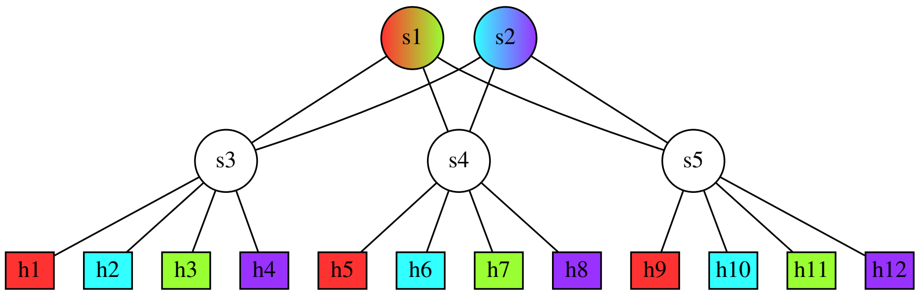

Figure 5 – Core switch allocation according to hosts VLANs.

2.2.3 Adaptive routing

The last policy that you will implement enables your SDN controller to adaptively send flows to the less loaded core router. Indeed, the switches will be continuously monitored to measure their available capacity regarding their current load. Then, new flows appearing in edge switches will be forwarded to the core switch having the less loaded down link.

In the topology illustrated in Figure 3, an example of such behavior would be if **h1** initiates a new HTTP connection towards **h7** while last measurements indicate that the path **s2–s1–s18–s5–s7** is at 6 Mbps while the path **s2–s1–s19–s5–s7** is at 1.5 Mbps. Then, your controller should lead **s1** to forward the packet to **s18** rather than **s19**. To perform the monitoring task, you are advised to use the **OFPPortStatsRequest** OpenFlow message which allows the controller to query a switch for port-related statistical information.

2.2.4 BONUS (2 points)

Bonus points will be given to students who implement, test and document in a proper manner an additional policy providing non-negligible improvements compare to the 3 policies presented before. Note that the reasons why the policy provide improvements should be argued in the report using any relevant measurement.

2.3 Practical details

The policies that you have implemented should be provided in files named as follow:

- **spanningtree.py** for the spanning tree-based policy.
- **vlans.py** for the VLAN-based policy;
- **adaptive.py** for the adaptive policy.
- **bonus.py** for the bonus policy.

Note that your policies should be working properly for any of the topologies that can be generated by FatTreeTopo.py (considering the hardware limit of the device it runs on).

In addition to that, provide any test scripts that enables you to perform the relevant measurements that you include in your report.

3 Evaluation

Assignment evaluation will be based on the following criteria:

- **Functionality**. Your three controllers must behave according to the corresponding policy. They should work for any topology generated from FatTreeTopo.py, not just the one in the example.
- **Coding style**. This assignment is not given in a programming course context. However, at this step of your studies, we assume you are able to provide elegant solutions to complex problems. The coding style (and, consequently, your solution elegance) will be part the grading.
- **Documentation**. In the fashion of other popular programming languages, Python comes with a powerful tool for documenting code: `Docstring`.[3] We ask you to fully document your code with respect to Docstring standards.
- **Report**. You must write a short report describing:
  - the general overview of your three solutions;
  - anything special you had to do to fulfil the requirements, which is not obvious from **documented** code;
  - how you tested your policies, and what you observed;
  - your feedback about this assignment:
    - Where were the main difficulties?
    - How many hours of work did it take (per student)?
    - What would you suggest to improve this assignment?

4 Submission

The submission of your project is subject to the following rules:

1. an archive named **Team_xx.zip** (where **xx** must be replaced by your TeamID), which contains all required files, must be uploaded on the submission platform (see `https://submit.montefiore.ulg.ac.be`)
2. the deadline is **December, 17th, 23h59** (hard deadline).

[3]See, for instance, `https://www.datacamp.com/community/tutorials/docstrings-python`

References

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, “A scalable, commodity data center network architecture,” *ACM SIGCOMM computer communication review*, vol. 38, no. 4, pp. 63–74, 2008.