CPSC425/525 Project A

Project A Due: 11:59 pm, Feb 28.

You are welcome to discuss your project with other students at the conceptual level. You may reuse algorithms from textbooks. You may possibly reuse functions from libraries, but you *must* check any packages/libraries that you plan to use with the TA's. You should also mark on all the places where you reuse functions/libraries from other places. Otherwise, each student is expected write *all* of his or her code independently. All programs will be subject to automated checking for similarity. Any cases of plagiarism will result in an F for the entire course. **If you have any questions about policies about academic integrity, please talk to the professor.**

Please note: you should expect to spend a significant amount of time on the class projects this semester, probably at least 20 hours or more when they're all put together (or that much per project if you're new to Python and shell and network programming). Please plan accordingly. If you leave all the work until the weekend before it is due you are unlikely to have a successful outcome.

1 Overview

The purpose of this project is to get some hands-on experience designing a congestion aware routing protocol for data centers by using Software Defined Networks. This homework will be done as two separate but related projects. First (Project A), you need to demonstrate that you are able to build a simple topology on top of Mininet and setup multipath routing. (Project A is very straightforward and will confirm that everyone is comfortable with the software environment.)

Related papers

FatTree, SIGCOMM'08

http://ccr.sigcomm.org/online/files/p63-alfares.pdf

2 How grading works?

Since automated tools grade your projects, you are expected to follow the requirements and specifications of the project as they are described. Failure to do so can affect your grade, so please make sure that your program works with the given commands.

3 Project A

This section of course project has two parts. In part one, you set up a small data center topology; in part two, you proceed by setting up routing rules on the switches of the topology. Please feel free to ask any questions that you may have on the piazza forum.

3.1 Part One

For this part, using the facilities provided by Mininet, we implement a fat-tree topology. We will test your code with different value of k (up to 8), so please make sure your code is generic to all k values. If you need to review fat-tree topology or Mininet, please refer to the slides or your notes. Below is the specification of your submission:

Host names and IPs There are $k^3/4$ hosts in a fat-tree topology. Please name the hosts hX, where X ranges from 1 to $k^3/4$. Also use 10.0.0.X as the IP address of the hX, Also, hosts should use the `eth0` interface, e.g., h11 will have one interface called `h11-eth0`.

File and class names For this part of the assignment, you are expected to submit one file: "fat_topo.py". The file contains **FatTopo** class that implements a fat-tree topology. We will verify your code with different k values by invoking:

```
bash> mn --controller=remote --custom fat_topo.py --topo fat_topo,k
```

where k should be replaced with a value. For example, to test $k = 4$, we will invoke:

```
bash> mn --controller=remote --custom fat_topo.py --topo fat_topo,4
```

As such, please make sure that Mininet can run your topology.

3.2 Part Two

3.2.1 A brief introduction on centralized management in SDN

One benefit of using a centralized controller is the flexibility in implmentation of routing protocols. One naive approach for implementing routing is to send every packet to the controller. The controller then decides whether to drop the packet or send it to its final destination. Since every packet is sent to the controller, a large amount of control plane bandwidth is required – at least as much as the ingress bandwidth of the network; as such, this approach is never used in practice. Another way to manage the network is to send the first packet of every flow to the controller. The controller then installs rules on data-plane which routes or drops packets from the same flow. After which, all the remaining packets of a flow are routed through these rules. This approach, which we discussed in Ethane, is known as *reactive* installation of rules. Finally, if the behaviour of the network is deterministic and the operating policy of the network is simple enough, it might be possible to *proactively* install rules in data-plane. These rules specify how every possible packet in the network should be routed.

3.2.2 Routing

You are required to use the Ryu controller and Mininet. The routing logic should be based on the header of each packet. In Ryu, make sure that you are specifying the ethernet type (eth_type) [1] as well as other header fields.

As we have studied, there are multiple paths between the hosts in different pods (e.g., 4 paths in a fat-tree topology with $k = 4$). In this project, we statically assign paths for different flows. You are free to decide how to assign paths. For example, you can just use only one core switch. Alternatively, to utilize the paths better, for a flow from host x to y, you can choose to split the path based on $y \bmod m$ where m is the number of possible next hops at a switch, or $(x + y) \bmod m$.

Implementing a simple Learning Controller would be sufficient for this part of the assignment.

3.3 Checking the connectivity of the nodes

In this Project A, we will verify your code by checking the connectivity of the nodes. We will generate packets between each pair of hosts, and verify every packet reaches its destination. To generate packets, we will use the *pingall* command in the mininet.

```
mininet> pingall
```

We will not test how you assign paths, i.e., as long as each pair is connected, you will get full credit even if you do not utilize all paths. Instead, you are required to describe how you assign the paths.

[1] You would need to specify the ethernet type so that the OpenFlow switch knows that it is matching for IP packets; the ethernet type of IPv4 packets is 0x0800.

3.4 Deliverables

You are expected to submit the following documents:

1. **Code:** Two files should be submitted, "fat_topo.py", which contains the Mininet topology, and "routing.py", which holds the routing code.
2. **README.txt:** In this file you should describe how you assign the paths.

Please make that all files are in the root of your git branch.

3.5 Submission Procedure: Github

As discussed in the class, you are expected to use Github to submit your project. Use the following Github Classroom link to create your assignment repository. You can then clone the repository to your VM by running:

```
bash> git clone git@github.com:Harvard-CS145/projecta-USERNAME.git
```

Afterwards, you can submit your files, e.g., the README.txt file, by using the following set of commands:

```
bash> git add README.txt # assuming README is in the root of your repository.
bash> git commit -m  "COMMIT MESSAGE" # please use a reasonable commit message,
               # especially if you are submitting code.
bash> git push origin master # Push the code to the remote repository.
```

In addition, please write your name in 'name.txt' and commit that change.

If you have any questions regarding the submission procedure, please do not hesitate to drop a line on Piazza, . [2]

4 Grading for Project A

The total grades for project A is 100 as follows:

- **10:** for your description of your routing in README.txt.
- **90:** we will test 3 values of k: 4, 6, 8. Each k has 30 scores. For each k, the score is proportional to the percentage of pairs that are connected. (30 means all can be connected). Your score will be halved if you write specific cases for each of $k = 4, 6, 8$.
- **deductions based on late policies**

[2] It might be wise to make sure that your repository is setup correctly by pushing an empty file to the repository a few weeks before the deadline.

5 Reference documents and files

5.1 Virtual Machine

For the purpose of this project, we have prepared a virtual machine that comes with Mininet, Ryu, and Scapy installed. You can use this virtual machine with VirtualBox, VMWare, KVM, etc. The virtual machine is available at:

`https://drive.google.com/file/d/1C4iMPl9sg0dYPd1q6zg6gGR9yiBtjMH4/view?usp=sharing`

The username and password to the virtual machine are: mininet/mininet

Finally, to connect to the virtual machine you can use the ssh command (or PuTTY, if you are on windows). By default we have mapped the port 1337 of the host machine to port 22 of the virtual machine, so you should be able to access the machine as such:

```
ssh mininet@localhost -p 1337
```

If you have trouble accessing the virtual machine, feel free to create a new post on Piazza.

5.2 Ryu Lists

Finally, Ryu comes with a very comprehensive list of tutorials and books. If at any point in the project you had trouble using the Ryu controller, we strongly suggest that you check out the following resources:

- http://ryu.readthedocs.org/en/latest/developing.html
- http://osrg.github.io/ryu-book/en/Ryubook.pdf
- https://github.com/osrg/ryu/tree/master/ryu/app