CSci551 Fall 2012 Project A

Assigned: Jan. 19, 2015. Project A Due: Feb 20.

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 Python packages that you plan to use with the TA's. You need to check allowed libraries in Project Information on class moodle. Otherwise, each student is expected write *all* of his or her code independently. All programs will be subject 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. That said, Project A is a "warm-up" project. Projects B and C are be much larger.

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 three 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 a dumb routing protocol. (Project A does not really evaluate anything advanced, but it should confirm that everyones on the same page and is comfortable with our software environment.) Project A has an early due date. Project A should be relatively short for students used to working under Linux and Python; if not, it should help get you up to speed.

In Project B you will implement an *arbiter*. We will supply traffic generators at end hosts, and you will program the communication between hosts and the arbiter application that you will write. The purpose is to rate limit traffic and select different paths for different traffic. It will probably be due just after the midterm. Project B will be much larger than Project A; you should plan your time accordingly. Project B will be assigned later and partially overlaps with Project A.

Project C will be assigned later. It will be smaller than Project B but bigger than Project A. It will build on Project B. Project C will involve extending your Project B arbiter

implementation with additional features, e.g., congestion aware routing and monitoring.

Related papers

FatTree, SIGCOMM'08

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

FastPass, SIGCOMM'14

http://dspace.mit.edu/handle/1721.1/88141

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 initial 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, and in part two, you proceed by setting up routing rules on the switches of your topology. Please feel free to ask us any questions that you may have on the forums or through email.

3.1 Part One

For this part, by using the facilities provided by Mininet, you will implement a fat-tree topology with ($k = 4$). If you need a review of fat-tree topology or Mininet, please refer to the slides and your notes. Below is the specification of your submission:

Host IPs There are 16 hosts in a fat-tree topology with $k = 4$. Please use 10.0.0.X as the IP address of the hosts, where X ranges from 1 to 16. Following this convention, all the hosts lie in the 10.0.0.0/27 subnet – 10.0.0.0 and 10.0.0.31 are reserved.

File and class names For this part of the assignment, you are expected to submit one file: "fat.py". In that file, a class called **FatTopo** exists that implements the fat-tree topology. We will verify your code by invoking:

```
mn --custom fat.py --topo fattopo
```

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

With the centralized controller, it is possible to perform routing in several ways. The naive approach is to send every packet to the controller. The controller can then decide whether to drop the packet or route it to its final destination. Since every packet is sent to the controller, a large amount of control plane bandwidth is required, as such, this approach is almost 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. Afterwards, all the packets in that flow use these rules, and are entirely routed in the data-plane. This approach, which we discussed in Ethane, is known as *reactive* installation of rules. Finally, if the behaviour of the network is known, and the operating policy of the network is simple enough, it might be possible to *proactively* install rules in data-plane that specify what should be done with each packet, i.e., no packets are sent to the controller.

In this project, since full connectivity between all host-pairs is desired, we can proactively install the rules, i.e., the controller can setup the routing when it boots up.

3.2.2 Routing

As discussed in the class, we will use source routing to route the packets throughout the network. Please recall that there are 4 paths between each host-pairs in a fat-tree topology with $k = 4$, you are free to choose any of these paths to route the packets between the two hosts. To communicate with other hosts, hosts attach VLAN-IDs to send packets toward its destination. We will propose a strawman approach later in this document.

A few notes:

- You are **required** to use the Ryu controller and Mininet.
- As you may recall, in a fat-tree topology, there are 4 unique paths between each pair of source and destination. For this part of the project, you are free to choose any of these paths to route packets.
- In a fat tree, there are 16 hosts, this translates to $16 * 15 = 240$ unique pairs. VLAN space is limited to 12 or 15 (with PCP) bits; this means that there is enough space to assign a separate VLAN for every pair.

The submission requirements are as follow:

1. **Controller program:** You are expected to use Ryu controller for this part of the project. Please use "source_routing.py" as the name of the file where your program lies. We will invoke your program by using the following commands:

```
ryu-manager source_routing.py &
mn --custom fat.py --topo fattopo --controller=remote,ip=127.0.0.1
```

2. **Questions:** Please include a "README.txt" file answering the following questions:

   (a) How many VLANs do you have in your entire network? Whats the minimum number of VLANs you need? Why?

   (b) Show the switch FIB table for a core, aggregator, and ToR switch. Then explain and describe how you planned to setup the FIB table, and how your routing works.

   (c) For a k=8 FatTree, how many hosts would we have? How many switches do we have? How many paths between a pair of hosts? How many VLANs do we need?

3.2.3 Strawman approach

Since our network has 16 hosts, there will be $16 * 15 = 240$ unique pairs of source and destinations. One way of setting up VLANs is to uniquely assign a VLAN-ID to each pair of source and destination. Using this scheme, whenever host A wants to send packets to host B, it uses a unique VLAN-ID that identifies the path that the packets should take to get to host B. The easiest way to generate this unique VLAN-ID is to shift the source-ID by 4 bits and add the destination-ID to it, e.g., if you have two hosts A (with $ID = 2$) and B (with $ID = 3$), the VLAN associated with the path from A to B is:

$$\text{VLAN}_{\text{From A to B}} = (2 << 4) + 3 = 35$$
$$\text{VLAN}_{\text{From B to A}} = (3 << 4) + 2 = 50$$

The rules on the switches will solely forward or drop packets based on these VLAN IDs. Please note that this approach is by no means the best approach, so feel free to let your imagination run wild.

3.3 Deliverables

You are expected to submit the following documents:

1. **Code:** Two files should be submitted, "fat.py", which contains the Mininet topology, and "source_routing.py", which holds the routing code.

2. **README.txt:** In this file you should answer the two questions in Section 3.2.2.

3.4 Submission Procedure: Github

As discussed in the class, you are expected to use Github to submit your project. For part A of the project, an empty repository has been created and associated with your Github username; you may clone the repository by using the following command:

```
git clone git@github.com:USC551Yu/USERNAME-projA.git
```

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

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

Please feel free to contact us on Piazza, if you had any questions regarding the submission procedure. [1]

[1] 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.