Ethane, Fastpass, and Preparation for Project

Omid Alipourfard

Outline: Part one, papers.

- Rethinking Enterprise Network Control

  Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Natasha Gude, Nick McKeown, and Scott Shenker.

- Fastpass: A Centralized "Zero-Queue" Datacenter Network

  Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal.

Outline: Part two, project and tools.

- Git and Github

  Creating a new repository.

  Cloning a repository.

  Pushing to repository.

  Checkout and branches.

  Stash, diff, log.

- Mininet

  Going through mininet tutorial (Part 1,3,4,5).

Outline: Part two, project and tools, contd.

- OpenFlow /SDN
  - Quick rundown
  - Rule format.
- Ryu Controller
  - Python syntax.
  - Going through mininet tutorial (Appendix).
- Fat-tree
  - Topology, and properties.

Outline: Part two, project and tools, contd.

- Routing

  VLANs, and some examples.

- Example

  Source routing on a small topology.

Ethane: Rethinking Enterprise Network Control

Fastpass: A Centralized "Zero-Queue" Datacenter Network

Git tutorial

Creating new repository

```
> mkdir csci551 && cd $_
  # Create a directory and change to it.
> git init .
  # Initialize Git.
> git add *
  # Add everything for commit
> git remote add git://github.com/SiGe/SomeProject
  # Add the github as a remote repository.
> git push origin master
  # Push your local content to the remote repository.
```

Cloning a new repository

```
> git clone https://github.com/SiGe/SomeProject.git
  # Clone the remote directory
> cd SomeProject
  # Go to the directory, and make changes.
> git commit -am “Made changes”
   # Commit the changes to the local branch.
> git push origin master
  # Push your changes to the remote repository -- if you have permission.
```

Pushing to repository

[Syntax]:

```
git push [alias] [branch]
# Push content to [alias] repository, from [branch]
# This command pushes from local:branch to alias:branch

git push [alias] [local-branch]:[remote-branch]
# You can change the remote branch by using the above command.
```

```
> git push origin master:my_branch
# Pushes the master branch to origin:my_branch
```

Checking out and Branches.

[Syntax]:

```
> git checkout [branch]
   # Changes the head to that branch, if the [branch] is available on origin
   # pulls that branch into [branch] and sets the upstream o
   # This command pushes from local:branch to alias:branch

> git checkout -b [branch]
   # Checkout [branch], and if it doesn't exist create [branch] using a
   # clone of the current branch.
```

Checking out and Branches, contd.

[Syntax]:

```
> git branch
   # List local branches
> git branch --remote
   # List remote branches
> git branch --all
   # List all branches
> git branch -d [branch]
   # Delete a branch
```

Mininet tutorial

Mininet

Mininet creates a realistic virtual network, running real kernel, switch and application code.

How to run?

```
> sudo mn --topo=<topology>
> sudo mn --topo=linear,3,1
  # would create 3 switches in a line with a
  # host connected to each switch
```

Mininet and Topology file

```
from mininet.topo import Topo
class YourCustomTopo(Topo):
    def __init__(self):
        Topo.__init__(self)
        # Create the custom topo here by using:
        # self.addHost, self.addSwitch, and self.addLink
    @classmethod
    def create(cls):
        return cls()
topos = {‘name_of_topo’: YourCustomTopo.create}
```

Mininet, contd.

Since we are running custom topology, we should pass our Python script which holds the topology. We are also using custom controller, so we should tell Mininet to use a remote controller (ours).

```
> sudo mn --custom=/path/to/custom.py
          --topo=name_of_topo
          --controller=remote
```

Mininet, contd.

Shows the list of available commands:

```
> help
```

Lists the topology of the network:

```
> net
```

Runs ping between all pairs of hosts (useful for debugging)

```
> pingall
```

Mininet, contd.

To run commands on specific hosts you can use the following syntax:

```
<host> <command>
```

e.g., to ping from h1 to h2:

```
> h1 ping h2
```

Similarly to run a python script, you could use:

```
> h1 python script.py
```

Mininet example

[checkout]: http://mininet.org/walkthrough/

OpenFlow tutorial

OpenFlow

1) Switches are dumb, and totally dependent on the controller.

2) Controller pushes rules to the switches, which in turn are used for packet forwarding. Controllers can tell the switches to perform a set of actions on packets that match specific patterns. These actions include:

```
drop, forward, modify header fields, push/pop vlan, etc.
```

You can see the list of available actions in OpenFlow specification.

OpenFlow, contd.

Rule format:

```
Rule: <priority> <match> <action> <counters>
```

Execution order:

Rules are run by in the descending order of their priority, i.e., rule with priority x runs sooner than rule with priority y, if x > y.

OpenFlow, contd.

An example of a switch rule table:

```
1000: match(proto=tcp)  -> actions(fwd=flood)

0200: match(ethtype=ip) -> actions(send_to_controller)

0000: match(*)          -> actions(drop)
```

OpenFlow and Mininet

You can use the following commands in Mininet to see the status of the switches and their forwarding tables:

List all the switches and their ports:

```
> dpctl show
```

List the flows on all the switches:

```
> dpctl dump-flows
```

OpenFlow specification

[Google]: OpenFlow specification, interesting sections:

6.1: Quick overview of the protocol and the messages.

Appendix: The OpenFlow Protocol

Ryu tutorial

Ryu

A framework for writing controllers that talk OpenFlow (v1.0,1.2.1.3,1.4)

You can think of it as a framework that makes it easy to write a controller like Ethane. If you don’t use it, you have to implement all the different parts of OpenFlow protocol, e.g., *hello, echo_request, echo_reply*.

Ryu - skeleton

```
from ryu.base import app_manager
from ryu.controller import ofp_event, dpset
from ryu.controller.handler import MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls

class Controller(app_manager.RyuApp):
    # the rest of the code
```

app_manager.RyuApp

```
from ryu.base import app_manager
```

You run your applications by using the following command:

```
> ryu-manager path_to_python_script
```

Inheriting from RyuApp makes your class compatible with ryu-manager.

Events

```
from ryu.controller import ofp_event, dpset
```

Ryu almost has events for everything that happens in the network:

- `dpset` module has events about the datapath elements (switches).
- `ofp_event` module has events about OpenFlow messages.

Dispatcher

```
from ryu.controller.handler import MAIN_DISPATCHER
```

Dispatchers are objects that when something happens on the network, Ryu would notify them (the dispatchers) and supply the event.

There are four dispatchers: `HANDSHAKE, CONFIG, MAIN, DEAD`

In this tutorial, we will be using `MAIN` exclusively, which usually is the dispatcher that you would use.

set_ev_cls

```
from ryu.controller.handler import set_ev_cls
```

set_ev_cls is a function decorator (a function that wraps another function).

You would use it to route specific events from dispatchers to the functions.

For example, if you are interested in OpenFlow PACKET_IN events, you would use:

```
@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
```

Ryu: an example application

[OpenFlow protocol files]

[Classes provided by Ryu for creating commands]

Flow modification on switches

```
from ryu.controller.handler import set_ev_cls
```

We would need to create a FlowMod message:

`ofp_parser.OFPFlowMod`

Specify that we want to add flows to the flow table:

`ofp.OFPFC_ADD`

And use the match and actions class to complete the FlowMod:

`ofp_parser.OFPMatch` and `ofp_parser.OFPActionOutput`