SDN controller clustering

Computer Networks module - SDN assignment

Michele Zanotti

Spring term 2018

Overview

In this paper are proposed four different lab activities meant to teach how to implement a cluster of controllers inside Mininet using some of the most common available methods. Each activity focuses on a single method and uses as reference the topology diagram showed in figure 1.

In particular, in each activity the following methods will be explained:

- **Activity 1**: implement a network with multiple local controllers using a python script and the middle-level Mininet API
- **Activity 2**: implement a network with multiple remote controllers using a python script and the middle-level Mininet API
- **Activity 3**: implement a network with multiple local controllers using a python script and the high-level Mininet API
- **Activity 4**: build a cluster of controllers using the tool *miniedit* provided by Mininet

The paper also includes a fifth lab activity, which is a challange meant to let the reader test the knowledge acquired with the execution of the previous four activities.

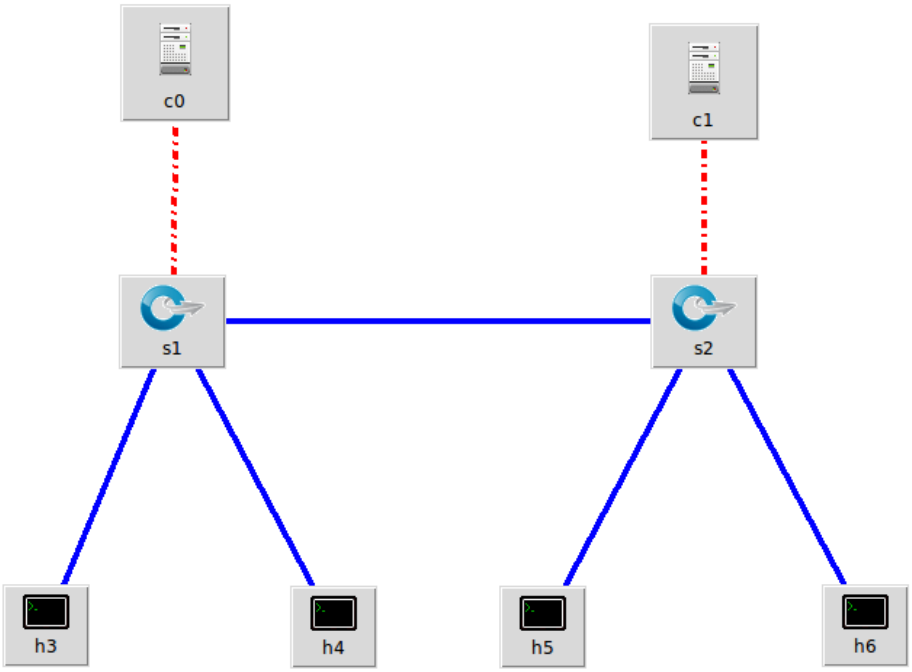

Figure 1: the simple topology that will be used as reference in the first four activities. The topology includes two switches, each one connected to a different controller. The two controllers *c0* and *c1* can be assumed local or remote depending on the lab activity.

Lab activity 1

Learning objectives

After finishing this lab activity you will be able to:

- Implement a cluster of local controllers inside Mininet using the Python middle-level Mininet API
- Test the network connectivity and the performance of a network which includes a cluster of local controllers
- Understand the main functions provided by the middle-level Mininet API required to implement a cluster of local controllers
- Reflect about the reasons of using more then one controller in SDN networks

Scenario

In this activity you will implement the simple topology shown in figure 1 using a Python script and the middle-level API provided by Mininet. The two controllers shown in the topology diagram will be local controllers for this activity. The topology has two different switches: each one will be connected to a different local controller.

Begin by creating a new Python script, then import Mininet classes required for this activity and define the function that will be used to create the topology. Inside the body of this function, create a new Mininet netowrk and add to it the required hosts, switches, links and controllers. After writing the script, execute it to create the network and test its connectivity and performance.

This lab activity assumes you are proficient in [...]. A basic knowledge of the Python programming language is also assumed.

Task 1: write the skeleton of the Python script

Step 1

Create a new Python script and edit with the text editor you prefer. If your editing it inside the Mininet virtual machine, it is suggested to use Vim text editor.

Step 2

Import the Python classes from the Mininet API:

```
#!/usr/bin/Python
from mininet.net import Mininet
from mininet.node import Controller, OVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info
```

Step 3

Make the script executable onyl as a program, set the verbosity level to "info" and call the function `multiControllerNet()`, which will be defined in the next step and will be used to create the required topology:

```
if __name__ == '__main__':
    setLogLevel( 'info' )
    multiControllerNet()
```

Step 4

Define the function that will be used to create the topology:

```
def multiControllerNet():
```

Step 6

Inside the body of the function `multiControllerNet()` create a new Mininet network:

```
net = Mininet( controller=Controller, switch=OVSSwitch )
```

The Mininet network is created invoking the Mininet constructor: the parameters passed to the constructor are the Controller class and the OVSSwitch class, therefore the Stanford/OpenFlow reference controllers and Open vSwitch switches will be used in the network we are goind to create. Note that these two classes are the default parameters in the Mininet constructor, so it is not really necessary to specify them.

Step 7

Save the text file as "*controllers-1.py*" in your custom directory inside the mininet virtual machine.

Task 2: add hosts to the network

Step 1

Inside the body of the function `multiControllerNet()` add the following line of code in order to print to the console that hosts are being created:

```
info( "*** Creating hosts \n" )
```

Step 2

Still inside the body of the function `multiControllerNet()`, create the four hosts required for the topology by adding them to the mininet network previously created:

```
h1 = net.addHost('h3')
h2 = net.addHost('h4')
h3 = net.addHost('h5')
h4 = net.addHost('h6')
```

The function used to add the hosts to the network is `addHost('name')`, which accept as parameter the name of the host that will be created. The hosts names in this network therefore will be `h3`, `h4`, `h5` and `h6`.

Task 3: add switches to the network

Step 1

Inside the body of the function `multiControllerNet()` add the following line of code in order to print to the console that switches are being created:

```
info( "*** Creating switches \n" )
```

Step 2

Still inside the body of the function `multiControllerNet()`, create the two switches required for the topology by adding them to the mininet network previously created:

```
s1 = net.addSwitch('s1')
s2 = net.addSwitch('s2')
```

Task 4: create links between nodes

Step 1

Inside the body of the function `multiControllerNet()` add the following line of code in order to print to the console that links are being created:

```
info( "*** Creating links \n" )
```

Step 2

Still inside the body of the function `multiControllerNet()`, create the links between the hosts and the switches and the link between the two switches:

```
net.addLink( h3, s1 )
net.addLink( h4, s1 )
net.addLink( h5, s2 )
net.addLink( h6, s2 )
net.addLink( s1, s2 )
```

Task 5: create the controllers

Step 1

Inside the body of the function `multiControllerNet()` add the following line of code in order to print to the console that reference controllers are being created:

```
info( "*** Creating (reference) controllers \n" )
```

Step 2

Still inside the body of the function `multiControllerNet()`, add to the Mininet network the two required controller:

```
c0 = net.addController( 'c0', port=6633 )
c1 = net.addController( 'c1', port=6634 )
```

The function use to create the controllers is `addController`, which accept as parameters the name of the controller which will be created and the TCP port that will be used by the switches for connecting to the controller.

Task 6: start the mininet network

Step 1

Append to the body the function `multiControllerNet()` the following line of code for printing to the console that the network is being started:

```
info( "*** Starting network \n" )
```

Step 2

Build the Mininet network:

```
net.build()
```

Step 3

Start the controllers:

```
c0.start()
c1.start()
```

Step 4

Start the switches, specifying for each switch the controller to wich connect:

```
s1.start( [ c0 ] )
s2.start( [ c1 ] )
```

Step 5

Start the Mininet CLI:

```
info( "*** Running CLI\n" )
CLI( net )
```

Step 6

Stop the network so that after the user exits the Mininet CLI the network is stopped:

```
info( "*** Stopping network\n" )
net.stop()
```

Task 7: execute the script and test the network

After finishing the task 6 the script for implementing the required topology is completed. The full script is shown in listing 1 at the bottom of this activity.

Step 1

Execute the script as root:

```
$ sudo python controllers1.py
```

Step 2

Test the created topology: verify the network connectivity between all hosts and the bandwidth between `h3` and `h6`. Write in the lines below the commands you used and the results you obtained.

Task 8: reflection

1 - What are the advantages of having more controllers instead of one single controller which serves all the switches of the network?

2 - Would the fault tolerance of the netowrk shown in figure 1 change if we used only one controller instead of two?

3 - How could we improve the fault tolerance of the netowrk shown in figure 1?

Listing 1: complete Python script required for Activity 1

```
#!/usr/bin/Python
from mininet.net import Mininet
from mininet.node import Controller, OVSSwitch
from mininet.cli import CLI
from mininet.log import setLogLevel, info

if __name__ == '__main__':
    setLogLevel( 'info' )
    multiControllerNet()

def multiControllerNet():
    net = Mininet( controller=Controller, switch=OVSSwitch )

    info( "*** Creating hosts\n" )
    h1 = net.addHost('h3')
    h2 = net.addHost('h4')
    h3 = net.addHost('h5')
    h4 = net.addHost('h6')

    info( "*** Creating switches\n" )
    s1 = net.addSwitch( 's1' )
    s2 = net.addSwitch( 's2' )

    info( "*** Creating links\n" )
    net.addLink( h3, s1 )
    net.addLink( h4, s1 )
    net.addLink( h5, s2 )
    net.addLink( h6, s2 )
    net.addLink( s1, s2 )

    info( "*** Creating (reference) controllers\n" )
    c0 = net.addController( 'c0', port=6633 )
    c1 = net.addController( 'c1', port=6634 )

    info( "*** Starting network\n" )
    net.build()
    c0.start()
    c1.start()
    s1.start( [ c0 ] )
    s2.start( [ c1 ] )

    info( "*** Running CLI\n" )
    CLI( net )

    info( "*** Stopping network\n" )
    net.stop()
```

Lab activity 2

Learning objectives

After finishing this lab activity you will be able to:

- Implement a cluster of remote controllers inside Mininet using the Python middle-level Mininet API
- Test the network connectivity and the performance of a network which includes a cluster of remote controllers
- Understand the main functions provided by the middle-level Mininet API required to implement a cluster of remote controllers

Scenario

In this activity you will implement the simple topology shown in figure 1 using a Python script and the middle-level API provided by Mininet. The two controllers shown in the topology diagram will be local controllers for this activity. The topology has two different switches: each one will be connected to a different local controller.

Begin by creating a new Python script, then import Mininet classes required for this activity and define the function that will be used to create the topology. Inside the body of this function, create a new Mininet netowrk and add to it the required hosts, switches, links and controllers. After writing the script, execute it to create the network and test its connectivity and performance.

This lab activity assumes you are proficient in [...]. A basic knowledge of the Python programming language is also assumed.