Réseaux : Configuration & Programmation

Introduction à la plateforme Mininet

Dino López

L'objectif de ce TP est de commencer à vous familiariser avec la plateforme Mininet. Mininet est un outil qui permet de déployer un réseau, composé de machines virtuelles (*VMs* par ses sicles en anglais) ou réelles, exécuter le vrai code des machines, utiliser des vrais commutateurs (matériel ou logiciel), à l'aide d'un nombre réduit de commandes.

Lorsque le réseau déployé est composé uniquement par de VMs et de commutateurs logiciel, elle pourrait être déployé à l'intérieur d'une seule machine (un ordinateur portable, par exemple).

Mininet utilise des technologies très avancés : Linux Containers et commutateurs SDN.

Exercice de programmation Python

```
public class JavaClassExample{
  private String name;

  public JavaClassExample(){
    name="Unconnu";
  }

  public void setName(String n){
    if ( n != null) {
        name = n;
    } else {
        return;
    }
  }
  public String getName(){
    return name;
  }

  public static void main(String args[]){
    JavaClassExample javaClassExample = new JavaClassExample();
    javaClassExample.setName("Visitor");
    System.out.println("Hello " + javaClassExample.getName());
  }
}
```

1. Traduisez le code ci-dessus en Python. La condition « `if ( n != null )` » peut être traduit comme « `if n` » en Python.

```
class PythonClassExample:
	_name = ""

	def __init__(self):
		self._name="Inconnu"

	def setName(self,n):
		if n:
			self._name = n
		else:
			return

	def getName(self):
		return self._name

if __name__ == '__main__':
	pythonClassExample = PythonClassExample()
	pythonClassExample.setName("Visitor")
	print "Hello " + pythonClassExample.getName()
```

2. Créez un script Python qui ouvre un fichier en mode écriture et lecture (ex. f = open("tt.txt","w+")) pour y garder ce que vous écrirez depuis le clavier. Lors que vous rentrerez uniquement le caractère entré '\n', le fichier doit être lu et vous devez afficher son contenu. Notez que vous pouvez lire un fichier ligne par line avec la méthode « readline() » ou toutes les lignes avec « readlines() », et y écrire avec la méthode « write() ». Notez aussi que le fichier STDIN est disponible par la clase stdin de la bibliothèque sys. Pour importer a bibliothèque sys, exécutez la ligne « import sys ».

```
def writeFile(f):
	while 1:
		line = sys.stdin.readline()
		if line == '\n':
			break
		else:
			f.write(line)
	f.flush()

def dumpFile(f):
	for line in f.readlines():
		print line.rstrip(); # Remove the last character of that line (i.e. \n)

def main():
	print "I'll create or rewrite file tt.txt"
	print "Introduce as many lines as you want to be stored in that file"
	print "To stop wirting to tt.txt, just enter an empty new line"
	f = open("tt.txt","w+")
	writeFile(f)
	print "the content of the file tt.txt is"
	print ""
	f.seek(0); # Go to the start of the file
	dumpFile(f)
```

```
if __name__ == '__main__':
        main()
```

Premier contacte avec Mininet

Les exercices ici proposés sont inspirés du tutoriel du site officiel de Mininet http://mininet.org/walkthrough/

Premièrement, retenez bien que vous devez exécuter Mininet avec les droits d'administrateur (i.e. avec la commande sudo).

```
$ sudo mn
```

Si tout s'est bien passé, à la fin vous devez voir l'invite de commande Mininet (le mode CLI de Mininet)

> Important : si lors d'un test, la topologie n'a pas été créée correctement, ou si Mininet a finis avec un code d'erreur, vous devez le « nettoyer » avec la commande
>
> ```
> $ sudo mn -c
> ```

« mininet> », et une topologie réseau devrait être créée. La topologie par défaut est appelé minimal.

Toutes les commandes disponibles dans la CLI mininet peuvent être liste avec la commande « help »

```
mininet> help
```

Les clients qui ont été créés dans notre réseau virtuel peuvent être listés avec la commande « nodes »

```
mininet> nodes
```

Par convention, les clients du réseau sont appelés h1, h2, … hX. Les switches (commutateurs) sont appelés s1, s2, … sX, et, à oublier pour l'instant, les contrôleurs des switches sont appelés c0, c1, … cX.

1. Combien de switches et clients sont disponibles dans votre réseau virtuel ?

    Un switch s1 et deux clients (h1 et h2)

Les informations liées à chaque nœud créé peuvent être listé avec la commande « dump ».

2. Décrivez ce que vous obtenez comme information des switches et clients du réseau.

    <Host h1: h1-eth0:10.0.0.1 pid=1242>
    <Host h2: h2-eth0:10.0.0.2 pid=1243>
    <OVSSwitch s1: lo:127.0.0.1,s1-eth1:None,s1-eth2:None pid=1247>
    <OVSController c0: 127.0.0.1:6633 pid=1234>
    2 hosts, on voit les @IP des hosts, PID des namespaces et un switch avec 2 interfaces

Les liens créés peuvent être listés avec la commande « net »

3. Dessinez la topologie créée.

h1 ----- s1 ------- h2

La puissance de Mininet réside, entre autres, sur le fait que les clients sont des images de la machine hôte (là où Mininet est installé) et ils ont accès à l'espace système de son utilisateur. Il est donc possible d'exécuter « à l'intérieur » des clients plusieurs commandes disponibles dans l'OS. Pour cela, vous devez précéder la commande par le nom de la machine virtuel qui l'exécutera. Par exemple, pour voir la configuration des interfaces réseaux disponibles dans le client h1, on exécute

```
mininet> h1 ifconfig –a
```

4. Ajoutez à votre dessin de la topologie réseau les adresses IP associées à chaque client, ainsi que le masque de réseau.

5. Le client h2 peut être atteint par le client h1 ? Vérifiez-le par la commande ping. Donnez l'instruction complète que vous avez utilisée.

mininet> h1 ping -c3 h2

Pour sortir de mininet, utilisez la commande « exit ».

Pour éviter d'indiquer à chaque fois qu'elle machine doit exécuter quelle commande, et si vous voulez travailler avec une console proche de celle des machines réelles, vous pouvez

1. Exécuter Mininet avec l'option « -x »
   `$ sudo mn –x`
2. Exécuter la commande xterm et indiquer le client à attacher.
   mininet> xterm h1

La première option ouvrira automatiquement une fenêtre xterm pour chaque client, switch et contrôleur présent dans la topologie.

D'autres topologies réseaux

Mininet fournit, en plus de la topologie « minimal », la topologie « single », « linear » et « tree ». Pour charger l'une de ces topologies, utilisez l'option « --topo ». Par exemple :

```
$ sudo mn --topo single
```

« single» tout court donne la même topologie que minimal, mais on peut ajouter également comme argument un chiffre, qui indique le nombre de clients à créer. Par exemple single,3.

1. Exécutez les commandes suivantes et dessinez la topologie créée :

1. $ sudo mn --topo linear

   H1 ---- s1 ---- s2 ---- H2

2. $ sudo mn --topo linear,3

```
h1 ---- s1 ---- s2 ---- s3 ---- h3
          |
          h2
```

3. $ sudo mn --topo tree

```
h1 ---- s1 ---- h2
```

4. $sudo mn --topo tree,2

```
h1 ---- s2 ---- s1 ---- s3 ---- h4
        |               |
       h2              h3
```

5. $sudo mn --topo tree,depth=2,fanout=3. À quoi servent les paramètres depth et fanout ?

```
            S1
          /   |  \
        S2    S3   S4
      / | \ / | \/ | \
     H1 ….
```

Topologies personnalisées

Et si on souhaitait travailler avec une topologie autre que *tree, linear* ou *simple* ? Bien, dans ce cas là, il faut créer une topologie personnalisée à l'aide de l'API Python de Mininet. Supposez que vous voulez créer un client h1 connecté directement au client h2 et ce dernier connecté à un switch s1. Voici le code Python à écrire

```python
from mininet.topo import Topo

class Test_Topo(Topo):
    def __init__(self):
        "Create P2P topology"

        # Initialize topology
        Topo.__init__(self)

        # Add hosts and switches
        h1 = self.addNode('h1')
        h2 = self.addNode('h2')
        s1 = self.addSwitch('s1')

        # Add links
        self.addLink(h1, s1)
        self.addLink(h2, s1)
```

```
topos = {
        'toptest': (lambda: Test_Topo())
}
```

1. Sauvegardez dans un fichier toptest.py le code précédent. Ensuite exécutez-le avec la commande

$ sudo mn --custom /chemin/vers/toptest.py --topo toptest

2. Si le réseau est créé correctement, ouvrez un terminal pour chaque client et explorez la configuration des cartes réseaux.

```
mininet> h1 ifconfig -a
mininet> h2 ifconfig -a
```

3. Vérifiez par la commande ping la connectivité entre les machines h1 et h2.

```
mininet> h1 ping -c3 10.0.0.2
```

Exécution automatiques de commandes

Pouvoir créer un réseau personnalisé est une bonne chose, mais devoir exécuter à chaque fois à la main les différents commandes pour exécuter nos expériences (par exemple, exécuter un ping pour vérifier la connectivité entre 2 machines), n'est pas très confortable.

Pour résoudre ce problème là, on utilisera encore l'API Python. Le code suivant (mais incomplet) exécute un ping depuis la machine h1 vers la machine h2. Comme vous pouvez l'apprécier, pour exécuter une commande quelconque sur une machine avec l'API, on fait appel à la méthode cmd() et on donne comme argument la commande à exécuter sous la forme d'une chaine de caractères.

```
from mininet.net import Mininet
from mininet.topo import Topo
from mininet.node import Node
from mininet.cli import CLI
from mininet.link import TCLink
from mininet.log import setLogLevel

class Test_Topo(Topo):
        …


topos = {
        'toptest': (lambda: Test_Topo())
}

def topTest():
        topo = Test_Topo()
        net = Mininet(topo=topo, link=TCLink)
        net.start()
```

```
        h2 = net.get('h2')
        h2int1 = h2.intf('h2-eth0')
        print h1.cmd('ping -c 3 %s' % h2.IP(h2int1))
        CLI(net) ; # sans cette ligne, on ne verrait jamais le CLI
        net.stop() ; # ne pas oublier de détruire le réseau

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

Cependant, notez bien que pour exécuter l'ensemble du code ajouté, on fera appelle directement à Python cette fois-ci.

$ sudo python /chemin/vers/fichier.py

Note : *Si lors de l'exécution de code Python, vous tombez sur un message qui indique que le contrôleur n'a pas pu être contacté, vérifiez tout d'abord que le fichier /usr/bin/ovs-testcontroller. Si ce n'est pas le cas, installez le package openvswitch-testcontroller (apt-get update; apt-get install openvswitch-testcontroller) afin d'installer sur votre machine le fichier précédemment recherché. Enfin, créez un lien, appelé « controller » vers la commande ovs-testcontroller (e.g. ln -s /usr/bin/ovs-testcontroller /usr/bin/controller).*

1. Lisez et analysez le code ci-dessus afin de trouver quel(s) ligne(s) de code doivent être ajouté(s) afin de pouvoir exécuter le programme correctement. Est-ce que le client h1 arrive à atteindre le client h2 ?

   Non, Il fallait obtenir une instance du client h1, afin de pouvoir exécuter h1.cmd() correctement.

2. Créez un script shell qui reçoit comme argument un chiffre entier N, et qui ensuite écrit N fois sur le terminal « Hello World X », où X est le nombre de fois que la chaine de caractères a été écrite.

   ```
   #!/bin/bash

   for ((i=0; i<$1; i++))
   do
           echo "Hello World $i"
   done
   ```

1. Faites que le host h1 exécute votre script shell automatiquement (i.e. depuis votre script python)

   h1.cmd("bash hello.sh")

Et c'est la fin

1. Dans un fichier testdel.py, créez la topologie suivante h1 --- s1 --- s2 --- h2. Ajoutez un délai de 10ms entre s1 et s2. Vérifiez par la commande ping (qui doit être lancé automatiquement) que le délai est bien respecté.

```
#!/usr/bin/env python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node
from mininet.link import TCLink
from mininet.log import setLogLevel
from mininet.cli import CLI

class TestDel(Topo):
        "Double switch connecting a sender and a receiver"
        def __init__(self):
                Topo.__init__(self)
                s1 = self.addSwitch('s1')
                s2 = self.addSwitch('s2')
                h1 = self.addNode('h1')
                h2 = self.addNode('h2')
                self.addLink(s1, h1)
                self.addLink(s1, s2, delay='10ms')
                self.addLink(s2, h2)

topos = {
        'testdel': (lambda: TestDel())
}

def constTopo():
        topo = TestDel()
        net = Mininet(topo=topo, link=TCLink)
        net.start()
        CLI(net)
        net.stop()

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

La commande ping devrait montrer un temps d'un peu plus de 20ms, ce qui correspond au temps pour que la requête ping arrive au destinataire, plus le temps pour que la réponse arrive à l'émetteur.

2. Copiez votre fichier testdel.py dans un fichier testbw.py afin de mettre un lien de 100Mbps de capacité entre s1 et s2. Ensuite, manuellement, sur h2 exécutez la commande « iperf -s -i 1 » et après sur h1, la commande « iperf -c 10.0.0.2 -t 10 ». Interprétez le résultat obtenu sur h2 et dites si la limite de bande passante a été respectez. Argumentez votre réponse.

```
#!/usr/bin/env python

from mininet.topo import Topo
from mininet.net import Mininet
from mininet.node import Node
from mininet.link import TCLink
from mininet.log import setLogLevel
```

```
from mininet.cli import CLI

class TestBw(Topo):
        "Double switch connecting a sender and a receiver"
        def __init__(self):
                Topo.__init__(self)
                s1 = self.addSwitch('s1')
                s2 = self.addSwitch('s2')
                h1 = self.addNode('h1')
                h2 = self.addNode('h2')
                self.addLink(s1, h1)
                self.addLink(s1, s2, bw=100)
                self.addLink(s2, h2)

topos = {
        'testbw': (lambda: TestBw())
}

def constTopo():
        topo = TestBw()
        net = Mininet(topo=topo, link=TCLink)
        net.start()
        CLI(net)
        net.stop()

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

Sur une machine bien dimensionné en mémoire et CPU, on aura un débit légèrement inférieur aux 100Mbps. Sans le paramètre bw=100 le débit observé par iperf est largement supérieur à 100Mbps.

3. si vous avez réussi l'exercice précédent, nous vous proposons d'expérimenter l'impact du trafic sur le délai observé entre 2 hosts. Pour cela, vous devez créer une topologie avec 2 hosts (h1 et h2) connectés vers un switch s1, d'autres 2 hosts (h3 et h4) connectés vers un switch s2 et une liaison entre ces 2 switches. La liaison entre les 2 switches doit avoir une limite de bande passante de 100Mbps, ainsi qu'un délai de propagation de 10ms. Ensuite, vous démarrez iperf pour effectuer un transfert entre h1 et h3. Ensuite, vous déterminez le délai avec ping entre h2 et h4. Effectuez à nouveau l'expérience ping, mais cette fois-ci sans le transfert entre h1 et h3. Commentez les résultats.

   Comportement attendu : si la bande passante de 100Mbps et le délai de propagation sont bien respectés, ping doit trouver un délai (temps) de 20ms, mais ce délai doit augmenter lors que le transfert avec iperf est en marche.

4. Augmentez maintenant la valeur du paramètre « -t » de iperf (testez plusieurs valeurs et utilisez des valeurs différentes pour h1-h3 et h2-h4) et baissez la valeur de la bande passante disponible. Commentez vos résultats. Présentez sous forme graphique le reporting de

l’application iperf en fonction des paramètres que vous faîtes varier (utilisez par exemple gnuplot).

Exercice entièrement à la charge de l’étudiant