Sommaire

1. Mise en place de l'environnement

Pour réaliser ce TP, Vous utiliserez la machine virtuelle précédemment configurée

1.1. Lancement des outils spécifiques

Pour la réalisation ce TP, nous allons utiliser un contrôleur SDN open-source nommé OpenDaylight. https://www.opendaylight.org En plus d'être libre et gratuit, il offre aussi un ensemble de logiciels permettant d'ajouter un certain nombre de fonctionnalités à ce dernier. La simulation du réseau se fera au travers de l'outil mininet http://mininet.org/.vue dans l'activité pratique précédente.

1.1.1. OpenDayLight

Je peux commencer par lancer mon OpenDayLight (contrôleur) avec la commande suivante :

```
sudo -E karaf
```

1.1.2. Mininet

Une fois le contrôleur lancé, il nous faut construire un réseau avec des switch supportant le protocole OpenFlow13. Pour cela, nous allons utiliser le logiciel mininet. Ce logiciel est normalement déjà installé sur votre machine virtuelle, si ce n'est pas le cas, installez-le depuis les dépôts. Une fois installé, vous pouvez commencer par supprimer les configurations préexistantes :

```
sudo mn -c
```

Sur ma machine mininet je définie une topologie :

```
sudo mn --topo linear,4,2 --mac --controller=remote,ip=192.168.177.132,port=6653 --switch ovs,protocols=OpenFlow13
```

topo : définit la topologie, avec 4 switch et 2hôtes par switch

controller : définit où est situé le contrôleur

switch : ovs, définit le type de switch que l'on veut utiliser, ici c'est des OpenVswitchs

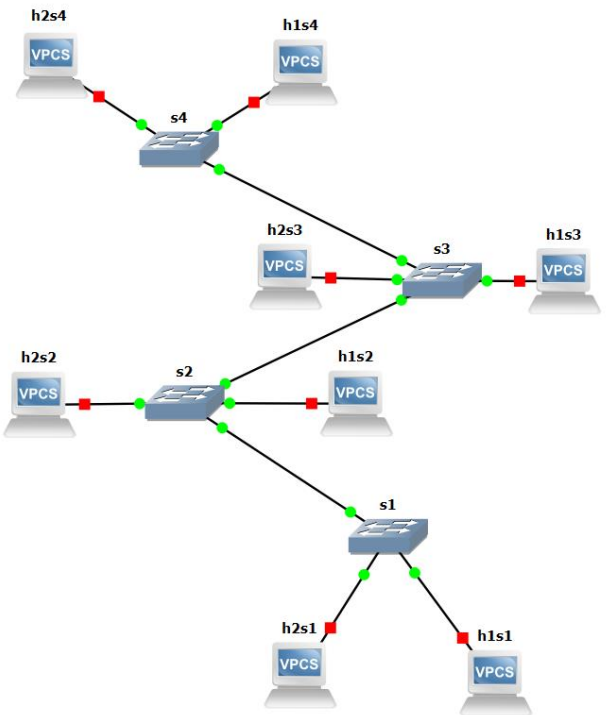
protocols : on lui spécifie le protocole, ici il faut bien faire attention à la version d'OpenFlow

mac : on va donner des adresses MAC aux hosts

Une fois dans l'interface avec la topologie chargée, vous pouvez interagir avec les switchs ainsi que les hôtes.

En vous basant uniquement sur les informations données par mininet faites l'exercice suivant :

1) Je dessine la topologie que mininet vient de me créer en spécifiant les noms à chaque fois :



2. Interaction entre le contrôleur et les Switch

Maintenant que mon architecture est mise en place je peu me connecter en interface WEB à ODL :

Pour accéder à mon contrôleur en interface WEB je peux me rendre à l'IP suivante :

http://192.168.1.200:8181/index.html#/login

Les logins sont : admin et admin

Quand je me rends dans l'onglet « topology » je me rend compte que seuls les switchs sont présents

Pour résoudre ce problème et faire apparaître les hôtes je peux lancer la commande suivante :

```
mininet> pingall
```

Je retourne maintenant dans le contrôleur et je rafraîchis la page. Je peux donc voir toute la topologie avec les hôtes présents

Pour transmettre et récupérer de l'information vers ou depuis les switchs, vous avez la possibilité d'utiliser :

- Yangam : qui a été installé depuis le contrôleur

- postman : application externe, permettant de forger des requêtes de type REST Dans un souci de lisibilité et de compréhension, nous allons utiliser postman.

3. Création des flux

3.1. Paramétrage de Postman

Postman est un outil qui va nous permettre de générer des requêtes http. Il va donc dans le cadre de ce TP se substituer à la couche application du modèle SDN.

Pour lancer Postman, j'utilise la commande postman dans le terminal et s'il n'est pas installé, je l'installe grâce à la commande suivante :

```
sudo snap install postman
```

Dans l'onglet Authorization, vérifiez que TYPE soit bien en Basic Auth. Pour valider ces étapes, nous allons définir une configuration basique pour un switch. Commençons par pousser une nouvelle table de flux. Pour cela, on doit spécifier la bonne url et la bonne action qui est PUT :

3.2. Création d'un premier flux simple

http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:1/table/0/flow/101

IP:PORT : celui du contrôleur, on se connecte du côté "nord"

restconf/config/opendaylight-inventory :nodes/node : définit la configuration que l'on veut manipuler/interroger

openflow:1 : définit le nom du switch sur lequel on veut effectuer l'action

table : on spécifie que l'on va manipuler une table spécifique

0 : précise le numéro de la table

flow/101 : on va manipuler le flux 101

Dans postman je place l'URL que je vais interroger (en mode PUT) puis je vais dans l'onglet « body »

Je passe en mode « raw » et je remplace le « text » par « XML »

Je place le code XML suivant dans le body :

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<flow xmlns="urn:opendaylight:flow:inventory">
  <id>101</id>
  <instructions>
    <instruction>
      <order>0</order>
      <apply-actions>
        <action>
          <order>1</order>
          <output-action>
            <output-node-connector>2</output-node-connector>
          </output-action>
        </action>
      </apply-actions>
    </instruction>
  </instructions>
  <table_id>0</table_id>
  <priority>100</priority>
  <flow-name>Port1surPort2</flow-name>
  <match>
    <in-port>openflow:1:1</in-port>
  </match>
</flow>
```

Maintenant je peux lancer la requête vers mon contrôleur

Pour vérifier que la table est bien présente sur le switch, il suffit de taper la commande suivante :

```
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s1

 cookie=0x2b00000000000003,     duration=3462.881s,     table=0,     n_packets=733,
n_bytes=62305, priority=100,dl_type=0x88cc actions=CONTROLLER:65535

 cookie=0x2b00000000000003,     duration=3458.917s,     table=0,     n_packets=50,
n_bytes=3668,   priority=2,in_port="s1-eth1"   actions=output:"s1-eth2",output:"s1-
eth3",CONTROLLER:65535

 cookie=0x2b00000000000004,     duration=3458.898s,     table=0,     n_packets=46,
n_bytes=3500,   priority=2,in_port="s1-eth2"   actions=output:"s1-eth1",output:"s1-
eth3",CONTROLLER:65535
```

```
 cookie=0x2b00000000000005,    duration=3458.898s,    table=0,    n_packets=295,
n_bytes=21798,  priority=2,in_port="s1-eth3"  actions=output:"s1-eth1",output:"s1-
eth2",CONTROLLER:65535

 cookie=0x0,     duration=252.339s,     table=0,     n_packets=0,     n_bytes=0,
priority=100,in_port="s1-eth1" actions=output:"s1-eth2"

 cookie=0x2b00000000000003, duration=3462.881s, table=0, n_packets=0, n_bytes=0,
priority=0 actions=drop
```

La nouvelle règle est bien apparue dans la bonne table

Pour m'assurer qu'elle est bien prise en compte je peux relancer un pingall dans ma machine mininet :

```
mininet> pingall

*** Ping: testing ping reachability

h1s1 -> X X X h2s1 X X X

h1s2 -> X h1s3 h1s4 h2s1 h2s2 h2s3 h2s4

h1s3 -> X h1s2 h1s4 h2s1 h2s2 h2s3 h2s4

h1s4 -> X h1s2 h1s3 h2s1 h2s2 h2s3 h2s4

h2s1 -> h1s1 h1s2 h1s3 h1s4 h2s2 h2s3 h2s4

h2s2 -> X h1s2 h1s3 h1s4 h2s1 h2s3 h2s4

h2s3 -> X h1s2 h1s3 h1s4 h2s1 h2s2 h2s4

h2s4 -> X h1s2 h1s3 h1s4 h2s1 h2s2 h2s3

*** Results: 21% dropped (44/56 received)
```

Les paquets sur el switch OpenFlow1 sont bien autorisés ou bloqués en fonction de la communication

Par exemple la machine h1s1 ne peut communiquer qu'avec la machine h2s1

Le résultat attendu est bien le bon. Les paquets sont autorisés ou non sur le Switch1 en fonction de la communication. Les ports d'un même switch peuvent être bloqués ou autorisé en fonction de ce que l'on veut

3.3. Création d'un flux bloquant par adresse MAC

Pour le blocage par adresse MAC je vais mettre en place les règles dans une autre table que la 0 (ici la table 2)

Je modifie l'URL :

http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:4/table/2/flow/126

Ensuite je modifie le code XML :

Le listing suivant définit l'action de dropper les paquets ayant une adresse MAC spécifique.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<flow xmlns="urn:opendaylight:flow:inventory">
  <strict>false</strict>
```

```
  <id>126</id>
  <instructions>
    <instruction>
      <order>0</order>
      <apply-actions>
        <action>
          <order>0</order>
          <drop-action/>
        </action>
      </apply-actions>
    </instruction>
  </instructions>
  <table_id>2</table_id>
  <priority>2</priority>
  <flow-name>BlocageMAC</flow-name>
  <match>
    <ethernet-match>
      <ethernet-source>
        <address>00:00:00:00:00:04</address>
      </ethernet-source>
    </ethernet-match>
  </match>
  <cookie_mask>255</cookie_mask>
  <installHw>false</installHw>
  <hard-timeout>1200</hard-timeout>
  <cookie>3</cookie>
  <idle-timeout>3400</idle-timeout>
  <barrier>false</barrier>
</flow>
```

Je vérifie maintenant avec la commande de tout à l'heure que la table a bien été prise en compte

```
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s4
 cookie=0x2b00000000000001,     duration=4829.908s,     table=0,     n_packets=1006,
n_bytes=85510, priority=100,dl_type=0x88cc actions=CONTROLLER:65535

 cookie=0x2b00000000000000,     duration=4825.946s,     table=0,     n_packets=70,
n_bytes=4956,   priority=2,in_port="s4-eth1"   actions=output:"s4-eth2",output:"s4-
eth3",CONTROLLER:65535

 cookie=0x2b00000000000001,     duration=4825.941s,     table=0,     n_packets=68,
n_bytes=4816,   priority=2,in_port="s4-eth2"   actions=output:"s4-eth1",output:"s4-
eth3",CONTROLLER:65535

 cookie=0x2b00000000000002,     duration=4825.936s,     table=0,     n_packets=396,
n_bytes=28056,   priority=2,in_port="s4-eth3"   actions=output:"s4-eth1",output:"s4-
eth2",CONTROLLER:65535

 cookie=0x2b00000000000001, duration=4829.908s, table=0, n_packets=0, n_bytes=0,
priority=0 actions=drop

 cookie=0x3, duration=73.377s, table=2, n_packets=0, n_bytes=0, idle_timeout=3400,
hard_timeout=1200, priority=2,dl_src=00:00:00:00:00:04 actions=drop
```

Je tente maintenant un ping entre l'hôte h1s4 et l'hôte h1s1 :

```
mininet> h1s4 ping h1s1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.4 icmp_seq=1 Destination Host Unreachable
...
```

Le ping ne fonctionne pas

Résolution 1 :

Pour résoudre ce problème je vais commencer par changer la priorité de la règle dans le code XML sans modifier la table :

```
<priority>50</priority>
```

Ensuite je vérifie que la modification a bien été prise en compte dans la table :

```
cookie=0x3, duration=81.343s, table=2, n_packets=0, n_bytes=0, idle_timeout=3400,
hard_timeout=1200, priority=50,dl_src=00:00:00:00:00:04 actions=drop
```

Je refais maintenant un ping entre les 2 hôtes :

```
mininet> h1s4 ping h1s1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.4 icmp_seq=1 Destination Host Unreachable
From 10.0.0.4 icmp_seq=2 Destination Host Unreachable
...
```

Le ping ne fonctionne toujours pas

Résolution 2 :

Pour résoudre le problème je vais repasser la priorité de ma règle à 2 mais changer de table (ici mettre 0) :

Code XML :

```
<table_id>0</table_id>
<priority>2</priority>
```

URL :

http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:4/table/0/flow/126

Je vérifie que la règle est bien prise en compte dans ma table 0 :

```
cookie=0x3, duration=5.940s, table=0, n_packets=0, n_bytes=0, idle_timeout=3400,
hard_timeout=1200, priority=2,dl_src=00:00:00:00:00:04 actions=drop
```

Je réessaye le ping entre mes hôtes :

```
mininet> h1s4 ping h1s1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.4 icmp_seq=1 Destination Host Unreachable
From 10.0.0.4 icmp_seq=2 Destination Host Unreachable
...
```

Le ping ne fonctionne toujours pas

Résolution 3 :

Pour résoudre le problème je vais augmenter la priorité de ma règle mais sans changer de table (je reste dans la table 0)

Code XML :

```
<table_id>0</table_id>
<priority>150</priority>
```

Vérification de la prise en compte de la règle dans la table :

```
cookie=0x3, duration=6.741s, table=0, n_packets=0, n_bytes=0, idle_timeout=3400,
hard_timeout=1200, priority=150,dl_src=00:00:00:00:00:04 actions=drop
```

Je peux réessayer un ping entre les hôtes :

```
mininet> h1s4 ping h1s1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.4 icmp_seq=1 Destination Host Unreachable
From 10.0.0.4 icmp_seq=2 Destination Host Unreachable
...
```

Le ping ne fonctionne pas

Pour conclure nous pouvons dire que l'action de la règle prend le dessus sur la priorité ou la table

Action > priorité > table

Peut importe la priorité ou la table, l'adresse MAC de la machine h1s4 est bloquée et ne pourra donc jamais ping une autre machine

Si nous avions une règle avec comme action « accept » et avec une priorité plus élevée que celle de l'action « drop » alors le flux passerait et l'hôte h1s4 pourrait ping les autres hôtes

De même pour la table. Si la table de la règle accept est supérieure à la table de la règle drop alors le flux passerait

3.4. Création d'un flux bloquant par adresse IP

Dans le prochain exemple nous allons créer une règle de drop par adresse IP sur le switch s2

URL :

http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:2/table/0/flow/10

Code XML :

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<flow xmlns="urn:opendaylight:flow:inventory">
  <strict>false</strict>
  <id>10</id>
  <instructions>
    <instruction>
      <order>0</order>
      <apply-actions>
        <action>
          <order>0</order>
          <drop-action/>
        </action>
      </apply-actions>
    </instruction>
  </instructions>
  <table_id>0</table_id>
  <priority>200</priority>
  <flow-name>BlocageIP</flow-name>
  <match>
    <ethernet-match>
      <ethernet-type>
        <type>2048</type>
      </ethernet-type>
    </ethernet-match>
    <ipv4-destination>10.0.0.2/32</ipv4-destination>
  </match>
  <cookie_mask>255</cookie_mask>
  <installHw>false</installHw>
  <hard-timeout>1200</hard-timeout>
  <cookie>1</cookie>
  <idle-timeout>3400</idle-timeout>
  <barrier>false</barrier>
</flow>
```

Vérification de la prise en compte de la règle dans la table du Switch s2 :

```
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s2

 cookie=0x1,      duration=70.718s,      table=0,      n_packets=2,      n_bytes=196,
idle_timeout=3400, hard_timeout=1200, priority=200,ip,nw_dst=10.0.0.2 actions=drop
```

Je peux maintenant essayer de ping la machine ayant l'IP 10.0.0.2 (ici h1s2) :

```
mininet> h2s3 ping h1s2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
15 packets transmitted, 0 received, 100% packet loss, time 14313ms
mininet> h1s4 ping h1s2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4078ms
```

Aucune machine ne peut atteindre la machine ayant l'IP 10.0.0.2 car les paquets sont drop par le switch

3.5. Création d'un flux bloquant sur de multiples critères

Dans ce dernier exemple nous allons mettre en place un flux bloquant sur de multiples critères

Ce flux sera mis en place sur le switch s4 dans la table 2

URL :

http://localhost:8181/restconf/config/opendaylight-inventory:nodes/node/openflow:4/table/2/flow/131

Code XML :

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<flow xmlns="urn:opendaylight:flow:inventory">
  <strict>false</strict>
  <id>131</id>
  <instructions>
    <instruction>
      <order>0</order>
      <apply-actions>
        <action>
          <order>0</order>
          <drop-action/>
        </action>
      </apply-actions>
    </instruction>
  </instructions>
  <table_id>2</table_id>
  <priority>2</priority>
  <flow-name>BlocageMulti</flow-name>
  <match>
```

```
    <ethernet-match>
      <ethernet-type>
        <type>2048</type>
      </ethernet-type>
      <ethernet-source>
        <address>00:00:00:00:00:02</address>
      </ethernet-source>
      <ethernet-destination>
        <address>00:00:00:00:00:06</address>
      </ethernet-destination>
    </ethernet-match>
    <ipv4-source>10.0.0.2/32</ipv4-source>
    <ipv4-destination>10.0.0.6/32</ipv4-destination>
    <ip-match>
        <ip-protocol>6</ip-protocol>
        <ip-dscp>2</ip-dscp>
        <ip-ecn>2</ip-ecn>
    </ip-match>
    <tcp-source-port>25364</tcp-source-port>
    <tcp-destination-port>8080</tcp-destination-port>
    <in-port>0</in-port>
  </match>
  <cookie_mask>255</cookie_mask>
  <installHw>false</installHw>
  <hard-timeout>1200</hard-timeout>
  <cookie>1</cookie>
  <idle-timeout>3400</idle-timeout>
  <barrier>false</barrier>
</flow>
```

Vérification de la prise en compte de la règle dans la table 2 du s4 :

```
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s4

cookie=0x1, duration=8.735s, table=2, n_packets=0, n_bytes=0, idle_timeout=3400,
hard_timeout=1200,
priority=2,tcp,in_port=0,dl_src=00:00:00:00:00:02,dl_dst=00:00:00:00:00:06,nw_src=
10.0.0.2,nw_dst=10.0.0.6,nw_tos=8,nw_ecn=2,tp_src=25364,tp_dst=8080 actions=drop
```

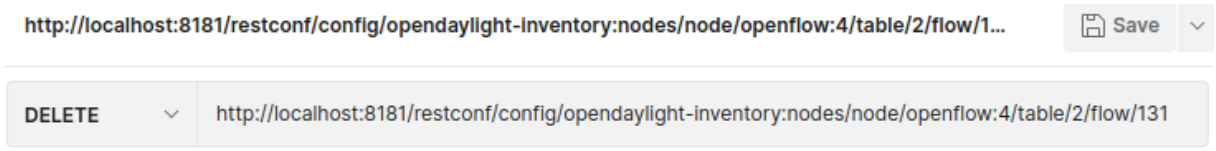
On ne peut pas vérifier dans mininet que cela fonctionne

3.6. Nettoyage des tables

Une fois une table dans un contrôleur, celle-ci reste tant qu'elle n'est pas supprimée du contrôleur. L'inconvénient majeur est qu'il est nécessaire de faire règle par règle. Pour cela, il vous suffit, à partir de PostMan de modifier le type de requête http en passant de PUT à DELETE et d'envoyer sur le contrôleur. Vous pouvez ensuite vous assurer que l'entrée dans la table a bien été supprimée en passant la commande 20.

Pour tester le bon fonctionnement de la suppression d'une règle je vais faire un test sur la dernière règle crée

Pour commencer je change le type de requête de « PUT » à « DELETE »



L'URL reste la même que pour la dernière règle ainsi que le contenu du fichier XML

Ensuite je peux lancer la requête afin de supprimer la règle de la table

La requête a bien été prise en compte

Je peux maintenant vérifier dans la table du switch s4 que la règle a bien disparue :

```
mininet> sh ovs-ofctl -O OpenFlow13 dump-flows s4
 cookie=0x2b00000000000001, duration=9387.765s, table=0, n_packets=1918, n_bytes=163030, priority=100,dl_type=0x88cc actions=CONTROLLER:65535
 cookie=0x2b00000000000000, duration=9383.803s, table=0, n_packets=94, n_bytes=6244, priority=2,in_port="s4-eth1" actions=output:"s4-eth2",output:"
s4-eth3",CONTROLLER:65535
 cookie=0x2b00000000000001, duration=9383.798s, table=0, n_packets=68, n_bytes=4816, priority=2,in_port="s4-eth2" actions=output:"s4-eth1",output:"
s4-eth3",CONTROLLER:65535
 cookie=0x2b00000000000002, duration=9383.793s, table=0, n_packets=418, n_bytes=29484, priority=2,in_port="s4-eth3" actions=output:"s4-eth1",output
:"s4-eth2",CONTROLLER:65535
 cookie=0x2b00000000000001, duration=9387.765s, table=0, n_packets=0, n_bytes=0, priority=0 actions=drop
```