

## TP SDN

---



**Stéphane Roche**  
**Abdelmajid Anka Soubaai**

*Département Génie Électrique et Informatique*  
*135, Avenue de Rangueil*  
*31077 Toulouse Cedex 4*

**Accès au sujet :** [tiny.cc/TP-SDN](https://tiny.cc/TP-SDN)

# Table of contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Prise en main de Floodlight et de Mininet / Containernet</b>	<b>3</b>
2.1	Démarrage du contrôleur SDN . . . . .	3
2.2	Émulation d'un réseau SDN . . . . .	3
2.3	Installation de règles de flux . . . . .	7
2.4	Interface Web du contrôleur Floodlight . . . . .	8
<b>3</b>	<b>Redirection transparente de paquets IP</b>	<b>11</b>
3.1	Personnalisation de topologie et lancement de serveurs Web . . . . .	11
3.2	Redirection transparente de paquets IP . . . . .	13
<b>4</b>	<b>Intégration de modules à Floodlight</b>	<b>17</b>
<b>5</b>	<b>Conclusion</b>	<b>18</b>

# 1 Introduction

Ce rapport retrace notre expérience au cours des TP consacrés à l'acquisition des compétences fondamentales dans le domaine des SDN. Ces TP représentent une opportunité unique de plonger dans l'univers dynamique des réseaux informatiques modernes. Notre objectif principal était de maîtriser les savoir-faire essentiels pour naviguer dans le monde en constante évolution des SDN, en utilisant des outils tels que mininet / Containernet et le contrôleur SDN Floodlight.

Au fil de ces Travaux Pratiques, on a cherché à développer une compréhension approfondie de plusieurs aspects clés. En premier lieu, l'utilisation de l'environnement d'émulation de réseau SDN, avec mininet / Containernet, nous a permis de simuler divers scénarios et de comprendre le fonctionnement des réseaux définis par logiciel. La manipulation du contrôleur SDN Floodlight a également été une étape cruciale, car cela nous a donné l'occasion de nous familiariser avec un outil couramment utilisé en production.

## 2 Prise en main de Floodlight et de Mininet / Containernet

### 2.1 Démarrage du contrôleur SDN

Nous débutons en premier lieu en lançant le contrôleur SDN à l'aide de la commande :

```
sdnvm@sdnvm:~/Téléchargements/Logiciels/SDNcontroller$ java -jar myTarget/sdncontroller.jar
2023-12-07 13:57:37.595 INFO [n.f.c.m.FloodlightModuleLoader] Loading modules from src/main/resources/floodlightdefault.properties
2023-12-07 13:57:38.222 WARN [n.f.r.RestApiServer] HTTPS disabled; HTTPS will not be used to connect to the REST API.
2023-12-07 13:57:38.223 WARN [n.f.r.RestApiServer] HTTP enabled; Allowing unsecure access to REST API on port 8080.
2023-12-07 13:57:38.223 WARN [n.f.r.RestApiServer] CORS access control allow All origins: true
```

La console affiche des messages signalant le lancement réussi des divers modules du contrôleur. Ainsi, le contrôleur est opérationnel et actif depuis un terminal. La phase suivante se déroule sur une autre fenêtre de terminal.

### 2.2 Émulation d'un réseau SDN

Afin d'émuler un réseau SDN, on utilise l'outil Mininet / Containernet via la commande "mn" (en sudo).

Nous allons utiliser essentiellement deux paramètres de cette commande :

- **-controller=remote,ip=A.B.C.D,port=6653** : Ce paramètre permet d'indiquer aux Switchs SDN créés par Mininet l'adresse IP du contrôleur à contacter et sur quel port le faire. Dans le cadre de ce TP, l'adresse IP du contrôleur est celle de la machine hôte égale à 10.0.2.15.
- **-topo=TOPO** : Ce paramètre permet de préciser le type de topologie réseau voulue. On peut utiliser des topologies prédéfinies par Mininet ou créer des topologies personnalisées (cf. deuxième partie). Dans les captures qui suivent, nous utilisons une topologie prédéfinie simple.

```

sdnvm@sdnvm:~$ mn --controller=remote,ip=10.0.2.15,port=6653
*** Mininet must run as root.
sdnvm@sdnvm:~$ sudo mn --controller=remote,ip=10.0.2.15,port=6653
[sudo] Mot de passe de sdnvm :
*** Creating network
*** Adding controller
Unable to contact the remote controller at 10.0.2.15:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1
*** Adding links:
(h1, s1) (h2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 1 switches
s1 ...
*** Starting CLI:
containernet> 

```

Ici, nous n'avons pas stipulé de topologie, nous avons donc obtenu une topologie par défaut composée de 2 hôtes connectés par 1 switch.

On peut relever sur la capture précédente que la connexion au contrôleur n'a pas été réalisée (erreur de manipulation avec démarrage du contrôleur après le lancement de Mininet).

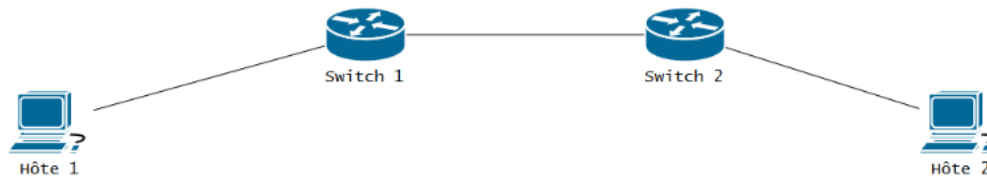
Nous essayons ensuite avec une topologie linéaire nous obtenons :

```

sdnvm@sdnvm:~$ sudo mn --controller=remote,ip=10.0.2.15,port=6653 --topo=linear,
2
*** Creating network
*** Adding controller
Unable to contact the remote controller at 10.0.2.15:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s2) (s2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
*** Starting CLI:
containernet> 

```

Nous obtenons la topologie attendue : 2 hôtes connectés via 2 switches.



Nous ajoutons ensuite un autre paramètre à notre commande , ce paramètre s'appelle **mac**.

Notre commande devient comme suite :

```
sdnvm@sdnvm:~$ sudo mn --controller=remote,ip=10.0.2.15,port=6653 --topo=linear,
2 --mac
*** Creating network
*** Adding controller
Unable to contact the remote controller at 10.0.2.15:6653
*** Adding hosts:
h1 h2
*** Adding switches:
s1 s2
*** Adding links:
(h1, s1) (h2, s2) (s2, s1)
*** Configuring hosts
h1 h2
*** Starting controller
c0
*** Starting 2 switches
s1 s2 ...
```

Le dernier paramètre “**mac**” permet d’assigner automatiquement aux différentes machines des adresses MAC à partir de la première adresse MAC disponible, c’est-à-dire 00:00:00:00:00:01.

**Note :** 2 commandes Mininet utiles sont **nodes** et **links** qui, respectivement, affichent les nœuds existants et les liens établis avec leur état de validité.

<pre>containernet&gt; nodes available nodes are: c0 h1 h2 s1 s2</pre>	<pre>containernet&gt; links h1-eth0&lt;-&gt;s1-eth1 (OK OK) h2-eth0&lt;-&gt;s2-eth1 (OK OK) s2-eth2&lt;-&gt;s1-eth2 (OK OK)</pre>
---	---

Pour se rendre compte de l'adressage MAC appliqué aux hôtes, on peut observer les paramètres réseau d'un hôte comme suit : **host ifconfig**.

De façon générale, **host "commande"** est compris par Mininet de sorte à ce que la commande soit exécutée par l'hôte déterminé.

```
containernet> h1 ifconfig
h1-eth0  Link encap:Ethernet  HWaddr 00:00:00:00:00:01
         inet adr:10.0.0.1  Bcast:10.255.255.255  Masque:255.0.0.0
         adr inet6: fe80::200:ff:fe00:1/64 Scope:Lien
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
         Packets reçus:26 erreurs:0 :0 overruns:0 frame:0
         TX packets:11 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 lg file transmission:1000
         Octets reçus:3065 (3.0 KB) Octets transmis:926 (926.0 B)
```

- On voit donc que l'adresse MAC associée à h1 est égale à celle du champ **HWaddr**.
- On peut aussi relever l'adresse IP via le champ **inet adr**

À présent, on peut tester la connectivité entre les hôtes en lançant un ping , comme ce qu'on a fait dans la figure ( essaie de ping h2 from h1) :

```
containernet> h1 ping -c 1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=229 ms

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 229.187/229.187/229.187/0.000 ms
containernet> █
```

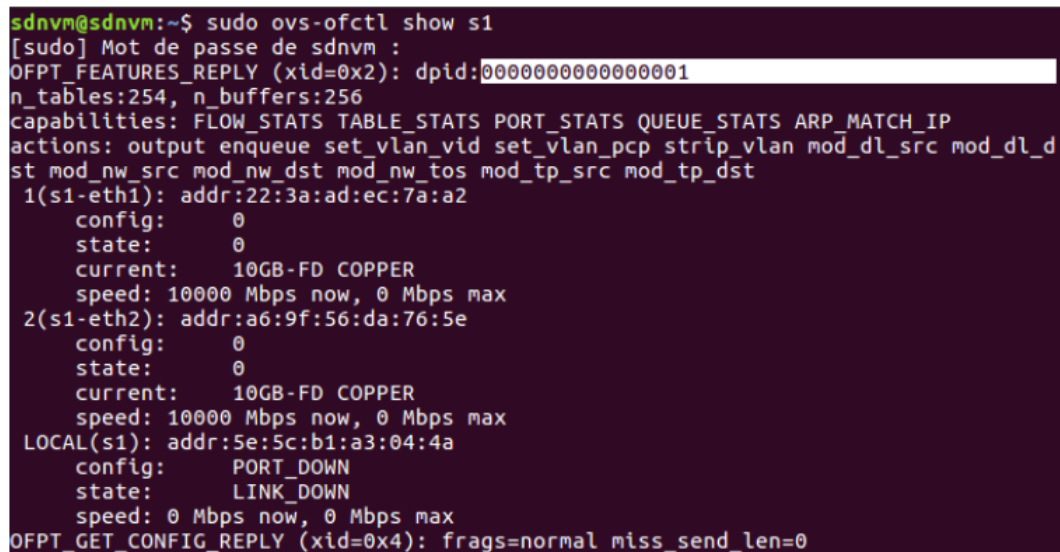
De cette manière, h1 effectue un unique echo request vers h2. On observe une réponse ce qui implique que la connectivité est OK. De plus, la résolution de nom logique h2 donne l'adresse IP 10.0.0.2 (et celle de h1 étant 10.0.0.1).

## 2.3 Installation de règles de flux

Le contrôleur SDN contient un module appelé **Static Entry Pusher** qui permet d'installer, de modifier et de supprimer des règles de flux sur les différents switchs sous contrôle. Ce module expose une API REST facilitant l'envoi des règles de flux aux différents switchs.

Pour installer une règle de flux, il nous faut plusieurs informations essentielles :

- Le switch SDN sur lequel la règle sera installée. Ici, la règle sera installée sur Switch 1. Il faut donc relever son **dpid** grâce à la commande "**sudo ovs-ofctl show s1** " dans un terminal en dehors de l'environnement Mininet.



```
sdnvm@sdnvm:~$ sudo ovs-ofctl show s1
[sudo] Mot de passe de sdnvm :
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst
mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
1(s1-eth1): addr:22:3a:ad:ec:7a:a2
  config: 0
  state: 0
  current: 10GB-FD COPPER
  speed: 10000 Mbps now, 0 Mbps max
2(s1-eth2): addr:a6:9f:56:da:76:5e
  config: 0
  state: 0
  current: 10GB-FD COPPER
  speed: 10000 Mbps now, 0 Mbps max
LOCAL(s1): addr:5e:5c:b1:a3:04:4a
  config: PORT_DOWN
  state: LINK_DOWN
  speed: 0 Mbps now, 0 Mbps max
OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

- Les **en-têtes** à lire pour identifier le flux : En spécifiant les adresses IP adéquates dans les champs "**ipv4-src**" et "**ipv4-dst**". Ici, il s'agit de supprimer tous les paquets provenant du Hôte 1 et à destination du Hôte 2 (donc **ipv4-src=10.0.0.1** et **ipv4-dst=10.0.0.2**).
- L'action à appliquer aux paquets appartenant à la règle de flux. En l'absence d'une action spécifiée, c'est l'action **DROP** qui est considérée par défaut.



commande finale est donc la suivante :

```
$ curl -X POST -d '{"switch":"00000000000000001", "name":"f1", "priority":"36000", "in_port":"1", "eth_type":"0x0800", "ipv4_src":"10.0.0.1", "ipv4_dst":"10.0.0.2", "actions":""}' http://10.0.2.15:8080/wm/staticflowpusher/json
```

```
sdnm@sdnm:~$ curl -X POST -d '{"switch":"1", "name":"f1", "priority":"36000", "in_port":"1", "eth_type":"0x0800", "ipv4_src":"10.0.0.1", "ipv4_dst":"10.0.0.2", "actions":""}' http://10.0.2.15:8080/wm/staticflowpusher/json
{"status" : "Entry pushed"}sdnm@sdnm:~$
```

Pour se rendre compte de la prise en compte de cette règle, nous pouvons tenter un nouveau ping de h1 vers h2.

```
containernet> h1 ping -c 1 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.

--- 10.0.0.2 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms

containernet> █
```

On peut observer que le ping depuis h1 n'obtient pas de réponses de h2. Le ping a été bloqué par la règle mise en place.

## 2.4 Interface Web du contrôleur Floodlight

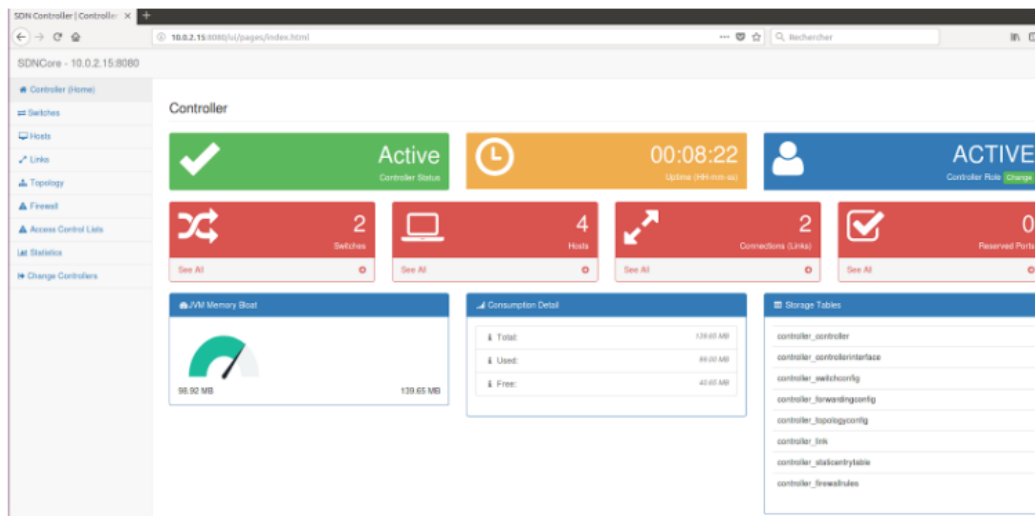
Le contrôleur dispose d'une interface Web consultable à partir de l'url : `http://10.0.2.15:8080/ui/index.html` (à l'intérieur de la VM).

À travers cette interface Web, on peut :

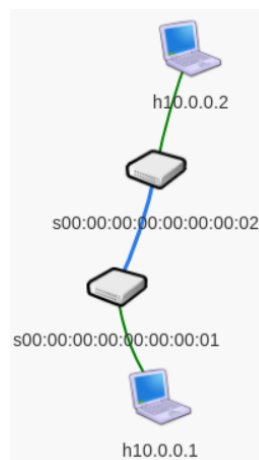
- Obtenir des informations (dpid, adresse MAC, adresse IP, ...) sur les switches, Obtenir des informations (adresse MAC, adresse IP, port, switch d'attache, ...) sur les machines hôtes
- Consulter les différentes tables de flux des différents switches
- Consulter la topologie

- Utiliser des modules applicatifs tels que le pare-feu, les ACLs (Access Control List), etc.

L'interface graphique obtenue depuis le contrôleur est la suivante :



D'après l'interface graphique du contrôleur on peut visualiser la topographie du réseau , elle est comme suite :



Depuis cette interface graphique on obtient :

- Des tables contenant les dpid et les adresses IP des switches déployés dans le réseau comme le montre la figure ci dessous :

Switches Connected		
Switch ID	IPv4 Address	Connected Since
00:00:00:00:00:00:01	/10.0.2.15:36704	Thu Dec 07 2023 13:35:18 GMT+0100 (CET)
00:00:00:00:00:00:02	/10.0.2.15:36706	Thu Dec 07 2023 13:35:18 GMT+0100 (CET)
Showing 1 to 2 of 2 entries		

Switch Roles	
Switch MAC	Role
00:00:00:00:00:00:02	MASTER
00:00:00:00:00:00:01	MASTER

- Une table contenant les les hôtes connectés dans le réseau avec leurs adresses IP, MAC ainsi que le switch et le port de celui-ci sur lequel l'hôte est connecté:

Hosts Connected					
MAC	IPv4 Address	IPv6 Address	Switch	Port	Last Seen
00:00:00:00:00:01	10.0.0.1	fe80::200:f5e00:1	00:00:00:00:00:00:01	1	1701956405616
00:00:00:00:00:02	10.0.0.2	fe80::200:f5e00:2	00:00:00:00:00:00:02	1	1701956421989
a6:83:ba:b2:6e:7c		fe80::a483:baf1:6b2:6e7c			1701956246901
f2:2b:c7:53:24:43		fe80::f02b:c78f:4e53:2443			1701956248229
Showing 1 to 4 of 4 entries					

- Une table contenant les liens internes (les liens transparents aux hosts entre les switches) avec les ports et les dpid des switches:

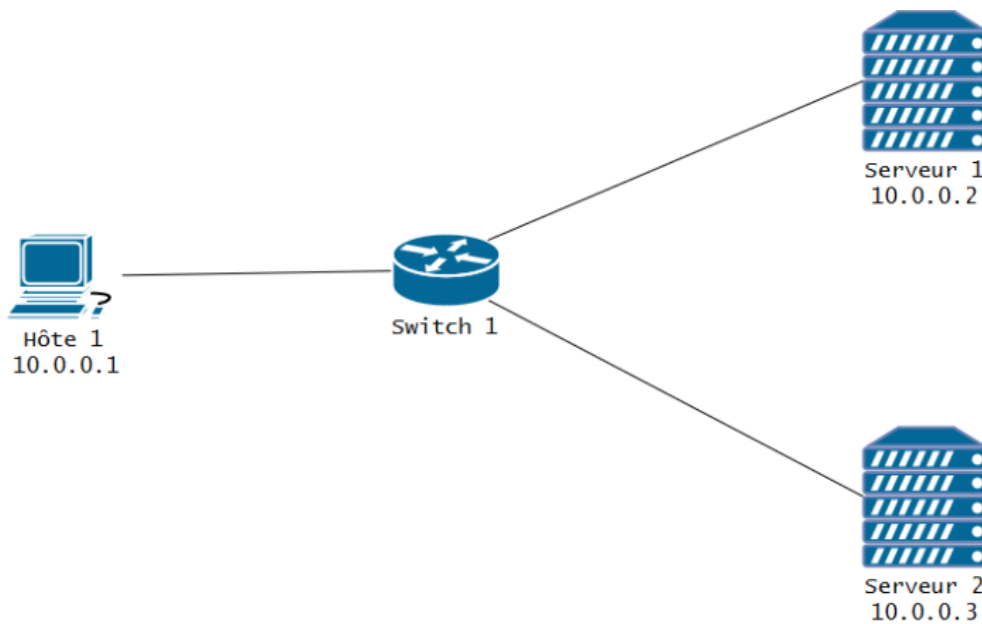
Internal Links						
Direction	Source Port	Source Switch	Destination Port	Destination Switch	Type	
bidirectional	2	00:00:00:00:00:00:01	2	00:00:00:00:00:00:02	internal	
Showing 1 to 1 of 1 entries						

## 3 Redirection transparente de paquets IP

### 3.1 Personnalisation de topologie et lancement de serveurs Web

Mininet ne créant pas de serveur Web par défaut, on définit une topologie comportant trois machines hôtes et un switch.

Ensuite, on lance un Serveur Web dans deux des machines hôtes créées par Mininet pour obtenir la topologie suivante :



Pour ce faire, nous utiliserons le fichier python (topo2.py) adéquat renseignant la topologie ci-dessus.

```
sdnvm@sdnvm:~/Téléchargements/Logiciels/SDNcontroller$ cd Topologies/  
sdnvm@sdnvm:~/Téléchargements/Logiciels/SDNcontroller/Topologies$ ls  
topo1.py topo2.py
```

Donc de la même manière que précédemment, on crée le réseau virtuel avec la topologie adéquate.

```
sdnvm@sdnvm:~/Téléchargements/Logiciels/SDNcontroller$ sudo mn --custom Topologi
es/topo2.py --topo=mytopo --controller=remote,ip=10.0.2.15,port=2253 --mac
[sudo] Mot de passe de sdnvm :
*** Creating network
*** Adding controller
Unable to contact the remote controller at 10.0.2.15:2253
*** Adding hosts:
Host1 Server1 Server2
*** Adding switches:
Switch1
*** Adding links:
(Host1, Switch1) (Switch1, Server1) (Switch1, Server2)
*** Configuring hosts
Host1 Server1 Server2
*** Starting controller
```

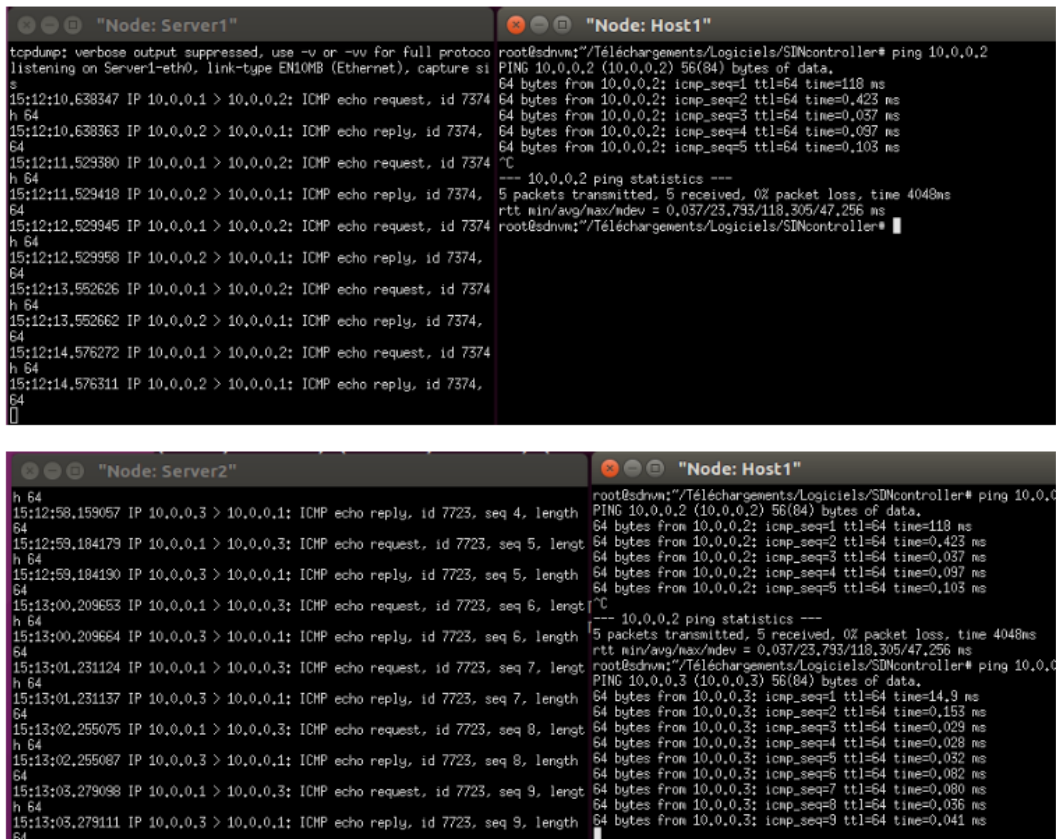
Puis on lance les serveurs sur les hôtes dédiés. Il s'agit de simples serveurs HTTP en écoute sur leur port 80 :

```
containernet> Server1 python -m SimpleHTTPServer 80 &
containernet>
containernet>
containernet>
containernet> Server2 python -m SimpleHTTPServer 80 &
containernet>
```

## 3.2 Redirection transparente de paquets IP

Tous les paquets IP en provenance de l'Hôte 1 et à destination du Serveur 1 doivent être orientés vers le Serveur 2 et ceci de façon transparente pour le client Hôte 1, c'est-à-dire ce dernier ne se rend pas compte de cette redirection et croit toujours communiquer avec le Serveur 1.

À cet effet, on utilise un émulateur de terminal compris dans Mininet, **xterm host** . Cela nous permettra via **tcpdump** d'observer les paquets que chaque appareil reçoit.



```
Node: Server1
tcpdump: verbose output suppressed, use -v or -vv for full protocol
listening on Server1-eth0, link-type EN10MB (Ethernet), capture size
15:12:10.638347 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 7374
h 64
15:12:10.638363 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7374,
h 64
15:12:11.529380 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 7374
h 64
15:12:11.529418 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7374,
h 64
15:12:12.529945 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 7374
h 64
15:12:12.529958 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7374,
h 64
15:12:13.952626 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 7374
h 64
15:12:13.952662 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7374,
h 64
15:12:14.576272 IP 10.0.0.1 > 10.0.0.2: ICMP echo request, id 7374
h 64
15:12:14.576311 IP 10.0.0.2 > 10.0.0.1: ICMP echo reply, id 7374,
h 64
0

Node: Host1
root@sdnvm:/Téléchargements/Logiciels/SINcontroller# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.118 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.423 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.037 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.097 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.103 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4048ms
rtt min/avg/max/mdev = 0.037/23.793/118.305/47.256 ms
root@sdnvm:/Téléchargements/Logiciels/SINcontroller#

Node: Server2
h 64
15:12:58.159057 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 7723, seq 4, length
64
15:12:59.184179 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 7723, seq 5, length
h 64
15:12:59.184190 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 7723, seq 5, length
h 64
15:13:00.209653 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 7723, seq 6, length
h 64
15:13:00.209664 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 7723, seq 6, length
h 64
15:13:01.231124 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 7723, seq 7, length
h 64
15:13:01.231137 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 7723, seq 7, length
h 64
15:13:02.255075 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 7723, seq 8, length
h 64
15:13:02.255087 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 7723, seq 8, length
h 64
15:13:03.279098 IP 10.0.0.1 > 10.0.0.3: ICMP echo request, id 7723, seq 9, length
h 64
15:13:03.279111 IP 10.0.0.3 > 10.0.0.1: ICMP echo reply, id 7723, seq 9, length
h 64

Node: Host1
root@sdnvm:/Téléchargements/Logiciels/SINcontroller# ping 10.0.0
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.118 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.423 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.037 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.097 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.103 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4048ms
rtt min/avg/max/mdev = 0.037/23.793/118.305/47.256 ms
root@sdnvm:/Téléchargements/Logiciels/SINcontroller# ping 10.0.0
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data:
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=14.9 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.153 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.028 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.028 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.032 ms
64 bytes from 10.0.0.3: icmp_seq=6 ttl=64 time=0.082 ms
64 bytes from 10.0.0.3: icmp_seq=7 ttl=64 time=0.080 ms
64 bytes from 10.0.0.3: icmp_seq=8 ttl=64 time=0.036 ms
64 bytes from 10.0.0.3: icmp_seq=9 ttl=64 time=0.041 ms
```

Par défaut, les pings fonctionnent normalement et les serveurs perçoivent les pings normalement. Maintenant, on va installer les règles de flux nécessaires pour établir une redirection transparente.

On peut donc récupérer le dpid du Switch 1 de deux manières :

- via le terminal

```
sdnvm@sdnvm:~/Téléchargements/Logiciels/SDNcontroller/Topologies$ sudo ovs-ofctl  
show Switch1  
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000000000000001
```

- Via l'interface Web à l'url 10.0.2.15:8080/ui/index.html

Switches Connected		
Switch ID	IPv4 Address	Connected Since
00:00:00:00:00:00:01	10.0.2.15:49576	Thu Dec 07 2023 14:09:43 GMT+0100 (CET)
Showing 1 to 1 of 1 entries		

Switch Roles	
Switch MAC	Role
00:00:00:00:00:00:01	MASTER

Pour ce qui est des en-têtes, il s'agit de capturer tous les paquets IP provenant de Host1 et à destination de Server1. On a donc besoin des adresses IP et MAC de Host1 et Server1.

On peut les obtenir grâce à l'interface Web du contrôleur :

Hosts Connected						
MAC	IPv4 Address	IPv6 Address	Switch	Port	Last Seen	
00:00:00:00:00:01	10.0.0.1	fe80::200:fe:00:01	00:00:00:00:00:01	1	1701958682992	
00:00:00:00:00:02	10.0.0.2	fe80::200:fe:00:02	00:00:00:00:00:01	2	1701958666609	
00:00:00:00:00:03	10.0.0.3	fe80::200:fe:00:03	00:00:00:00:00:01	3	1701958732143	
Showing 1 to 3 of 3 entries						

Pour ce qui est des autres champs d'en-tête :

- Paquets de type IP : le champ “eth-type” vaut 0x0800.
- Les paquets provenant de Host1 : les champs à utiliser sont “eth-src” et “ipv4-src” de valeurs respectives “00:00:00:00:00:01” et “10.0.0.1”.
- Il faudra aussi préciser le port par lequel les paquets arrivent au switch. Il s'agit du port 1 (port connecté au Host1).

Comme il s'agit d'une redirection vers Server2, il faut une réécriture des champs "eth-dst", "ipv4-dst" prenant les valeurs respectives "00:00:00:00:00:03" et "10.0.0.3". Il faut également préciser le port de sortie avec output. Ici, c'est le port 3 (port connecté au Server2).

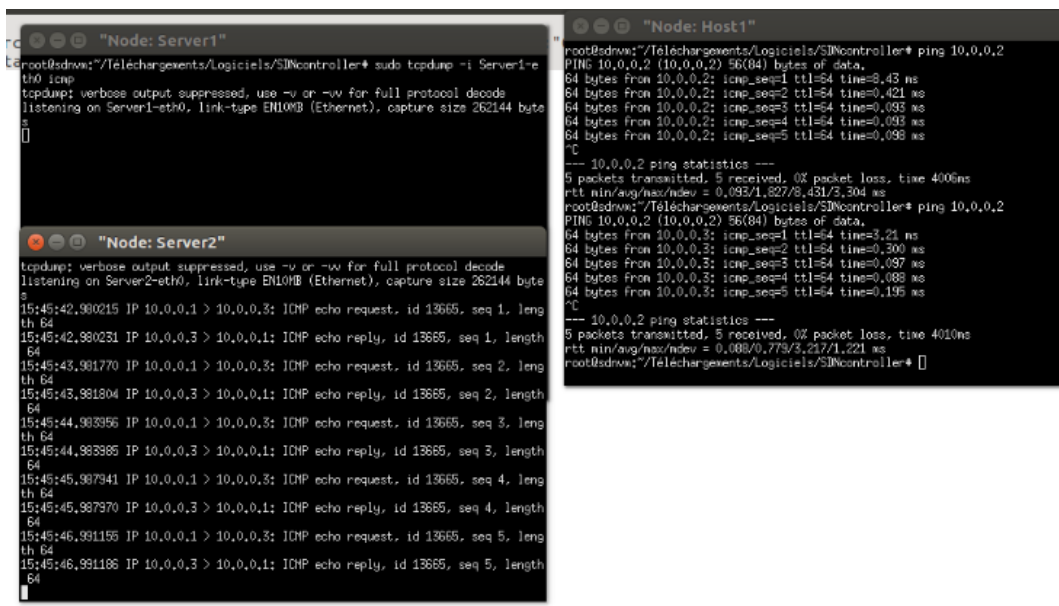
Pour pousser la règle à présent, la commande à utiliser est la suivante :

```
$ curl -X POST -d '{"switch":"00:00:00:00:00:00:00:01", "name":"f2",  
"priority":"36000", "in_port":"1", "eth_type":"0x0800",  
"eth_src":"00:00:00:00:00:01",  
"ipv4_src":"10.0.0.1", "eth_dst":"00:00:00:00:00:02",  
"ipv4_dst":"10.0.0.2",  
"actions":"set_field=eth_dst->00:00:00:00:00:03,set_field=ipv4_dst->  
10.0.0.3,output=3"}' http://10.0.2.15:8080/wm/staticflowpusher/json
```

Cependant il ne s'agit de la règle que dans un sens (Host vers Server), on peut établir une règle pour le retour avec un raisonnement analogue.

```
$ curl -X POST -d '{"switch":"00:00:00:00:00:00:00:01", "name":"f3",  
"priority":"36000", "in_port":"3", "eth_type":"0x0800",  
"eth_src":"00:00:00:00:00:03",  
"ipv4_src":"10.0.0.3", "eth_dst":"00:00:00:00:00:01",  
"ipv4_dst":"10.0.0.1",  
"actions":"set_field=eth_src->00:00:00:00:00:02,set_field=ipv4_src->  
10.0.0.2,output=1"}' http://10.0.2.15:8080/wm/staticflowpusher/json
```

Cette règle permet donc d'assurer le retour du paquet avec des champs semblables à ce qu'aurait eu Host1 sans déviation de trafic. Pour l'exemple qui suit, nous avons sciemment omis cette règle et voici ce que l'on obtient avec un nouveau ping :



```
root@sdnwm:/Téléchargements/Logiciels/SIMncontroller# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.43 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.421 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.093 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.093 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.098 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/ndev = 0.093/1.827/8.431/3.304 ms
root@sdnwm:/Téléchargements/Logiciels/SIMncontroller# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data:
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=3.21 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=0.300 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=0.097 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=0.088 ms
64 bytes from 10.0.0.3: icmp_seq=5 ttl=64 time=0.195 ms
^C
--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4010ms
rtt min/avg/max/ndev = 0.088/0.779/3.217/1.221 ms
root@sdnwm:/Téléchargements/Logiciels/SIMncontroller#
```



Host1 effectue un ping sur l'adresse 10.0.0.2 (Server1), cependant le tcpdump sur Server1 ne révèle aucun paquet reçu sur l'interface. En revanche, le tcpdump effectué sur Server2 révèle du trafic sur l'interface à destination de Server2 (lui-même).

D'autre part, il est intéressant de relever que Host1 reçoit un retour de 10.0.0.3 alors qu'il ping l'adresse 10.0.0.2. La redirection grâce à la première règle est donc valide.

Voilà pourquoi la seconde règle proposée juste avant la dernière capture permettrait d'assurer une transparence complète pour cette redirection.

## 4 Intégration de modules à Floodlight

L'objectif de cette partie est de développer un des modules suivants :

- Un redirecteur de paquets IP dans le temps: pendant une période T1, les paquets du client vont vers le Server1, et durant une deuxième période T2, les paquets vont vers le Server2. Ce comportement est répété indéfiniment.
- Un redirecteur de paquets IP à la demande: à la demande de l'utilisateur, et à travers un appel REST, les paquets sont dirigés vers le serveur 1 ou 2:
  - Exemple d'appel : un message POST avec le contenu "1" ou "2" vers `http://controlleur/api/module/serveur`
- Un moniteur de flux à la demande: faire la matrice de trafic par protocole. Exemple:
  - L'installation d'une surveillance se fera par un appel REST : un message POST avec le contenu "1" ou "0" vers `http://controlleur/api/module/monitor/<proto>`
  - Récupérer les informations de monitoring par un appel REST: message GET vers: `http://controlleur/api/module/monitor/<proto>`

Nous avons débuté le 2ème module, cependant nous n'avons pas été en mesure d'établir l'API REST comme indiqué. Nous n'avons donc pas pu compléter cette partie mais nous avons pu comprendre le principe de modules via Floodlight afin d'apporter une meilleure flexibilité à la gestion du réseau.

## 5 Conclusion

Ce compte rendu reflète notre expérience au cours de ces Travaux Pratiques dédiés à l’acquisition de compétences fondamentales dans le domaine des Réseaux Définis par Logiciel (SDN). Ces TP ont constitué une opportunité privilégiée pour plonger dans l’univers dynamique des réseaux informatiques modernes, avec pour objectif principal de maîtriser les savoir-faire essentiels nécessaires à l’évolution constante des SDN, en exploitant des outils tels que Mininet / Containernet et le contrôleur SDN Floodlight.

Au fil de ces travaux, nous avons cherché à développer une compréhension approfondie de plusieurs aspects clés. L’utilisation de l’environnement d’émulation de réseau SDN avec Mininet / Containernet nous a permis de simuler divers scénarios et de saisir le fonctionnement des réseaux définis par logiciel. La manipulation du contrôleur SDN Floodlight a constitué une étape cruciale, offrant une familiarisation avec un outil couramment déployé en production.

La mise en œuvre concrète a débuté par le lancement du contrôleur SDN à l’aide de la commande spécifiée. La console a confirmé le démarrage réussi des différents modules du contrôleur, le rendant ainsi opérationnel et accessible depuis un terminal. Les phases suivantes ont été consacrées à l’émulation d’un réseau SDN en utilisant Mininet / Containernet, avec une attention particulière portée aux paramètres de la commande et à la personnalisation de la topologie réseau.

L’expérience s’est enrichie avec la redirection transparente de paquets IP, impliquant la personnalisation de la topologie et le déploiement de serveurs Web dans un environnement contrôlé. Ces étapes ont été facilitées par l’utilisation de commandes Mininet spécifiques et l’ajout de paramètres pour atteindre les objectifs définis.