Faculté des Sciences & Techniques — Université de Limoges

Master 1ère année

Admin Rés

TP n°1

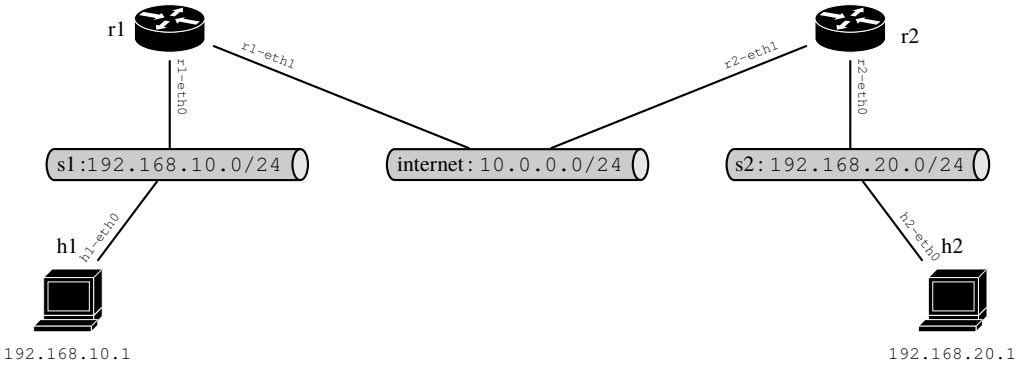
Configuration réseau, « sniffing » & virtualisation

Plateforme virtuelle

On va construire un réseau fermé, c-à-d sans accès vers Internet, composé de :

- 3 réseaux correspondant à 3 switches : `s1`, `s2` et `internet` ;
- deux routeurs `r1` et `r2` ;
- deux hôtes, `h1` et `h2`, chacun connecté dans le réseau `s` respectif.

switch/réseau	adresse
s1	192.168.10.0/24
s2	192.168.20.0/24
internet	10.0.0.0/24



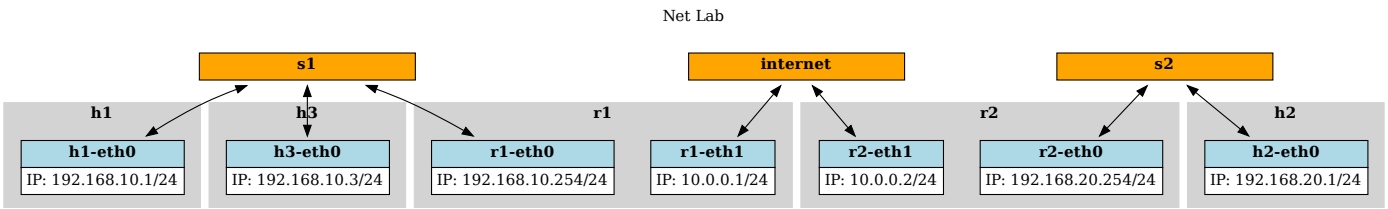
Création des fichiers de configuration de la simulation sur une VM avec `virtualbox` :
Vous installerez une VM debian/Ubuntu avec le paquet openvswitch-switch installé

Ces fichiers sont récupérables depuis le dépôt `git.p-fb.net` à l'aide de la commande suivante **exécutée** depuis la VM :

xterm
```
$ git clone https://git.p-fb.net/PeFClic/netlab.git
```

Le graphe suivant est obtenu automatiquement à partir du fichier de description « `build_architecture` » à l'aide du script Python `network_graph.py` :

xterm
```
$ python3 network_graph.py > graph.dot; dot -Tpng graph.dot -o graph.png
```



Travail à réaliser : routage

1 – a. Vous lancerez le script de **création** de notre « Lab » :

xterm
```
$ sudo ./build_architecture
```

Les switches étant persistants (il résiste à un redémarrage), il est possible que certaines commandes indiquent que "cela existe déjà" ce qui n'est pas grave.

Pour exécuter une commande dans un « *netns* »

Il faut indiquer le nom du « *netns* », suivi de la commande à exécuter :

xterm
```
$ sudo ip netns exec h1 ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
9: h1-eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UP mode DEFAULT group default qlen 1000
    link/ether f6:15:3c:c4:aa:07 brd ff:ff:ff:ff:ff:ff link-netnsid 0
```

*L'usage du `sudo` est **obligatoire**, car les netns créent de « l'isolation » et exigent des droits d'administrateur pour en franchir la séparation (passer du netns root à un autre netns).*
Vous pouvez également exécuter un shell dans un netns pour que toutes les commandes lancées depuis lui-même soient exécutées dans ce netns :

xterm
```
$ sudo ip netns exec h1 sudo -u rezo bash
```

l'ajout de `sudo -u rezo` permet de limiter les droits de ce shell à l'utilisateur `rezo`.

Si vous lancez une commande interactive dans un netns (qui ne rend pas la main immédiatement), **vous aurez besoin d'une nouvelle fenêtre**/nouveau shell pour exécuter une autre commande.

b. Dans chacun des « netns », exécutez les commandes suivantes :
☐ `ip link` ☐ `ip address` ☐ `ip route`
Vérifiez que la configuration de chaque netns est conforme au synopsis réseau.

c. vous testerez le routage :
- avec la commande `ping` :

  xterm
  ```
  $ sudo ip netns exec h1 ping -c 1 192.168.20.1
  ```
- avec la commande `traceroute` :

  xterm
  ```
  sudo ip netns exec h1 traceroute 192.168.20.1
  ```

  Vous recommencerez cette commande tout en interceptant le trafic sur r1 :

  xterm
  ```
  $ sudo ip netns exec r1 tcpdump -lnvv -i r1-eth1
  ```

  Pour n'intercepter que le trafic en « *time exceeded* » :

  xterm
  ```
  $ sudo ip netns exec r1 tcpdump -lnvv -i r1-eth1 | grep time
  ```

  Quel matériel répond à la commande « traceroute » ? Est-ce normal ?

2 – Vous **intercepterez le trafic** sur le réseau « internet » :

xterm
```
$ sudo ip netns exec r1 tcpdump -nvvl -i r1-eth1
```

et lancerez un ping de h1 vers h2 :

xterm
```
$ sudo ip netns exec h1 ping -c 1 192.168.20.1
```

Est-ce que l'on observe des messages ARP ? Pourquoi ?
Que retourne la commande suivante :

xterm
```
$ sudo ip netns exec r1 ip neighbor
```

Est-ce cohérent avec la trafic intercepté par tcpdump ?

3 – Vous testerez maintenant la **connectivité TCP** :

a. avec la commande `socat` en mode client sur h1 et en mode serveur sur h2 :

xterm
```
$ sudo ip netns exec h2 socat tcp-listen:6789 -
```

xterm
```
$ sudo ip netns exec h1 socat - tcp:192.168.20.1:6789
```

xterm
```
$ sudo ip netns exec r1 tcpdump -nvvl -i r1-eth0 'tcp and port 6789'
```

Comment se passe la connexion TCP ? Comment identifier le paquet contenant les données envoyées par socat si l'on saisit un texte au clavier ?

b. sans quitter la connexion (ou en la relançant au besoin), vous pouvez consulter l'état de la pile TCP/IP avec la commande `conntrack` :

xterm
```
$ sudo ip netns exec h1 conntrack -L
```

Est-ce que l'information est différente sur h1 ou h2 ? Pourquoi ?
Si vous fermez « brutalement » la connexion avec un « ctrl-c », est-ce que vous voyez un « TIME_WAIT » ? Pourquoi ?

c. Si vous lancez l'outil `nmap` depuis h1 vers h2 :

xterm
```
$ sudo ip netns exec h1 nmap 192.168.20.1
```

Que retourne-t-il ? Est-ce normal ?

4 – Vous testerez maintenant le **trafic http** :

xterm
```
$ sudo ip netns exec h2 python3 -m http.server 80
```

Puis à l'aide de la commande `curl` :

xterm
```
$ sudo ip netns exec h1 curl -v http://192.168.20.1/
```

Que retourne maintenant la commande `nmap` de h1 vers h2 ?

Pouvez vous utiliser le navigateur Firefox pour accéder au serveur Web :

xterm
```
$ ip netns exec h1 firefox
```

Quelle URL devez vous entrer ?

Vous testerez l'interception de trafic avec l'outil `tshark` :

xterm
```
$ sudo ip netns exec r1 tshark -i r1-eth1 -o "ip.use_geoip:FALSE" -l -Y
"http.response or http.request" -T fields -e ip.host -e tcp.port -e
http.request.full_uri -e http.request.method -e http.response.code -e
http.response.phrase -e http.content_length -e data -e text
```

Que pouvez vous en dire ? Par rapport à tcpdump ?

5 – Vous testerez le **débit** entre h1 et h2 :

xterm
```
$ sudo ip netns exec h2 iperf -s
```

Pour quitter le serveur, il suffit d'utiliser un « ctrl-c ».

xterm
```
$ sudo ip netns exec h1 iperf -c 192.168.20.1
```

Quel débit observez vous ? Pourquoi ? (indice : regardez dans le fichier créant le « lab »).