

Université			
-------------------	--	--	--

28 janvier 2024

		de Strasbourg
--	--	----------------------

Rapport projet

Réseaux Programmables RAPACE

Aurélia Bausch

Arnaud Filippi

Félix Lusseau

Sommaire

Introduction

Topologie

Implémentations des équipements

Méta-contrôleur, API et métrologie

Test d'une configuration

Pistes d'améliorations

Conclusion

I – Introduction

L'équipe a réalisé ce projet en essayant de le pousser jusqu'au bout. Le choix de topologie a été implémenté, les équipements requis avec leurs options également ainsi qu'un en bonus. Le méta-controller et l'API et la métrologie ont été presque totalement implémentés. Nous expliquerons plus loin nos difficultés rencontrées et nos choix en conséquence.

IMPORTANT:

La VM P4 fournie pour ce projet n'était pas fonctionnelle chez 2 des 3 membres du groupe. Ainsi nous avons procédé à une installation locale de P4. Il s'avère que la VM était un peu ancienne comparée aux dernières versions utilisées pour une installation locale de certains packages. Ainsi il faudra utiliser la dernière VM QEMU de l'ETH Zurich en Ubuntu 20.04 (<https://nsg-ethz.github.io/p4-utils/installation.html#use-our-preconfigured-vm>) pour que notre projet fonctionne à cause de la version de Python installée.

La procédure de lancement est décrite dans le README.md. Il faut notamment installer 2 dépendances Python nécessaires au méta-contrôleur avec `sudo pip3 install -r requirements.txt`. Elles doivent être installées en sudo car le méta-contrôleur doit être exécuté en tant que root afin de pouvoir lancer Mininet.

Enfin, il y a une typographie à respecter pour les équipements. Le nom d'un host commence toujours par un 'h' suivi d'un nombre : h1, h2, h40 valides. H1ost, host1 pas valides. De même pour un switch: s1, s2, s40 valides. Switch1, s1routeur pas valides.

Lien vers le GitHub : <https://github.com/FelixLusseau/RAPACE>

II – Topologie

L'utilisateur avant de démarrer le réseau doit décrire celui-ci au travers du fichier `network.yaml` contenant la liste des switches avec leur type d'équipement, la liste des hôtes et des liens souhaités.

Les équipements disponibles sont exactement :

- router
- router_lw
- load_balancer
- firewall

```
! network.yaml
1  RAPACE:
2  Switches:
3    s0: load_balancer
4    s1: firewall
5    s2: firewall
6    s3: load_balancer
7  Hosts:
8    h1:
9    h2:
10 Links:
11   - [h1, s0]
12   - [s0, s1]
13   - [s0, s2]
14   - [s3, s1]
15   - [s3, s2]
16   - [s3, h2]
```

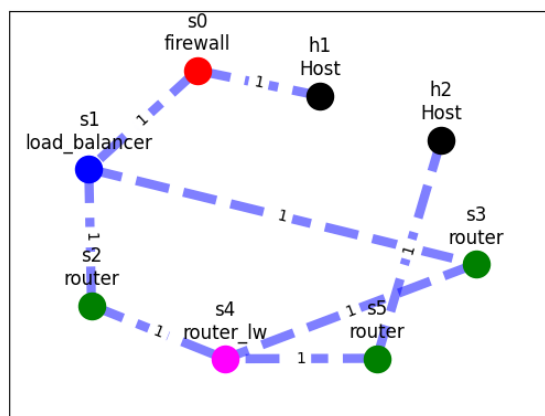






Figure 1 – Initialisation d'une topologie

Ici (Figure 1) la topologie décrite à gauche nous permet d'obtenir le graphe à droite.

Ensuite afin d'instancier le réseau il faut lancer le méta-contrôleur avec la commande :

```
felix.lusseau in  lr-server in RAPACE on  main [  ] via  v3.10.12
> sudo python3 meta_controller.py
```

On laisse ensuite le temps au réseau de s'instancier... et voilà ! La topologie est fonctionnelle. Il est possible de consulter le fichier `network.png` pour afficher le graphe (Figure 1) et un résumé est affiché dans le terminal. L'ensemble du réseau physique est une clique, mais le réseau logique est lui créé sur la topologie voulue par l'utilisateur.

III - Implémentations des équipements

L'ensemble des équipements étant fonctionnels nous allons ici juste décrire comment ceux-ci peuvent être manipulés et comment ils sont conçus. Les commandes présentées ci-dessous doivent être lancées dans le méta-contrôleur.

1) Firewall

Lorsque le firewall est instancié celui-ci va chercher les interfaces des deux équipements qui sont reliés à lui. L'utilisateur veillera à uniquement relier deux appareils au firewall.

Il peuple ensuite ses tables pour qu'un paquet reçu sur un port puisse être renvoyé sur l'autre si celui-ci n'est pas considéré comme indésirable par une règle ajoutée par l'utilisateur dans la table "fw". Nous avons pris la liberté de ne pas considérer le port source car celui-ci est aléatoire et change à chaque exécution d'une communication. Il est alors pratiquement impossible de bloquer un flux le matchant.

Une métadonnée utilisateur a été ajoutée afin d'y placer le port TCP ou UDP selon la situation afin de ne pas avoir une table TCP, une UDP et une dernière ICMP.

Le firewall compte le nombre de paquets filtrés par chaque règle. Il compte également le nombre de paquets reçus. Ceux-ci peuvent être affichés avec les commandes `see filters` qui affiche également la règle associée et `see load` respectivement.

Les commandes propres au firewall sont les suivantes et s'appliquent à tous les firewalls :

- `add_fw_rule <flow>`

La commande permet d'ajouter une règle de firewall. Flow est ici de la forme : "IP_src IP_dst port_dst protocol". Le protocole doit être spécifié en minuscules parmi TCP, UDP et ICMP (où le port n'est alors pas pris en compte).

- `see filters`

Pour chaque firewall, affiche le nombre de paquets filtrés par chaque règle présente sur le firewall.

2) Loadbalancer

Lorsque le loadbalancer est instancié, celui-ci attendra que l'utilisateur spécifie en ligne de commande le port "in". Pour cela il devra utiliser la commande "set_port_in" qui sera décrite juste en dessous. Une fois le port "in" choisi, l'ensemble des autres ports sont considérés comme "out"

Une fois le port "in" instancié, le loadbalancer va peupler ses tables.

Le loadbalancer instancie un meter initialement configuré pour limiter la charge par lien à 1 paquet par seconde (précisons que chaque lien possède son propre meter). Si le meter remarque un dépassement les paquets sont droppés. (On aurait pu imaginer une redirection sur un autre lien mais cela pose d'autres soucis comme une congestion ailleurs).

Le loadbalancer est flow aware. Pour un même ensemble protocole, d'adresses IP de destination, source, ports source et destination, le flux empruntera le même lien.

Le loadbalancer compte également le nombre de paquet reçu.

Les commandes propres au loadbalancer sont les suivantes :

- `set_port_in <load_balancer_name> <port_in> :`

Cette commande permet de changer le port_in (ou de l'instancier). Par exemple en reprenant la figure1 : `"set_port_in s1 s0"` instancie le port de l'interface entre le firewall et le loadbalancer en "in". Les interfaces avec les routeurs sont alors considérées comme out.

- `set_rate_lb <load_balancer_name> <rate> :`

Change la limite de paquet par seconde pour chaque lien pour le loadbalancer désigné, avec "rate" en paquet par seconde (par défaut fixé à un)

- `see rate :`

Affiche la valeur des limiteurs pour chaque loadbalancer.

Détails :

Pour le loadbalancer, l'implémentation se fait surtout du côté du contrôleur. Il faut faire attention à ce que la répartition des ports out soit aléatoire et complète. Un paquet provenant d'un port "out" match la table qui lui renvoie immédiatement une action pour effectuer le prochain saut sur le port "in". Un port provenant du port "in" passera alors par une table de hachage renvoyant un nombre aléatoire compris entre 0 et le nombre de port "out" moins 1. Il faut donc bien veiller à une bonne gestion des tables. Chaque paquet avant d'être envoyé passe par le meter propre à son lien. Si celui-ci dépasse la valeur limite, il marque le paquet et celui-ci sera plus tard drop.

3) Router

Lors de son instanciation, le routeur commence par calculer les plus courts chemins en prenant en compte les poids des liens vers l'ensemble des hôtes en Longest Prefix Match (LPM) ainsi que vers les Loopback des autres routeurs qui ont été ajoutées à la topologie lors de sa génération. Les IPs de Loopback sont en 10.100.0.x/32 où x est le numéro du routeur. Le routeur complète également la table "encap_routing" qui à un numéro de routeur donné lui associe le port de sortie et la MAC du routeur suivant pour l'atteindre. Cette table sera utilisée pour le routage lors de l'encapsulation.

Les routeurs sont basés sur le code du TP simple-routing de P4-Learning. Ils disposent donc de l'ECMP en cas de chemins de coûts égaux (avec et sans encapsulation). La table principale est "ipv4_lpm" qui associe un préfixe à un port de sortie en LPM.

Les routeurs supportent également l'encapsulation de paquets suivant le principe du Segment Routing. Seule la fonctionnalité de routeur de destination a été implémentée. Il est difficile dans la clique physique de déterminer à l'avance quels ports de sortie seront utilisés afin d'imposer le passage par un lien au paquet.

Tout d'abord, un utilisateur peut définir une règle d'encapsulation à l'aide de la commande ``add_encap_node <start_node> <flow> <end_node>`` où le flux ici n'est constitué que de l'IP source et l'IP de destination pour des questions de simplification. Tout le trafic entre les deux IPs sera alors encapsulé du nœud de départ au nœud d'arrivée. La règle directe est appliquée au routeur de départ et la règle inverse est appliquée au routeur de destination afin que le flux soit encapsulé à l'aller ET au retour (dans ce cas l'utilisation des ports pose problème car on ne pourrait pas encapsuler le flux retour en raison du choix aléatoire du port source).

La règle d'encapsulation est stockée dans la table `'encap_rules'` qui n'est consultée que si le paquet entrant n'est pas déjà encapsulé.

Le paquet doit matcher les deux IPs pour déclencher une règle et il en résultera l'ajout du header `segRoute_t` qui contient uniquement un entier définissant le routeur à atteindre avant désencapsulation et le type Ethernet est placé à `TYPE_SEGROUTE = 0x1234`.

On notera que les nœuds source et destination doivent être des routeurs et non low-energy (ceci est vérifié par le meta-controller).

En second lieu, si lors du calcul des plus courts chemins le next-hop est un `router_lw`, le routeur va ajouter une entrée dans la table `"encap_lw"` qui à un préfixe IPv4 associe un numéro de routeur comme point de passage (checkpoint). Celui-ci est déterminé en recherchant le premier routeur non low-energy sur le plus court chemin. Les routeurs low-energy ne comprenant pas l'IPv4, tout le trafic à destination des hôtes placés derrière doit être encapsulé pour les traverser, sans intervention de l'utilisateur.

Déroulement de l'encapsulation :

- Un paquet entrant matche soit une règle d'encapsulation utilisateur d'un flux ou bien une règle d'encapsulation due à la présence de routeurs low-energy. Le header `segRoute_t` est inséré et le type Ethernet passé à `TYPE_SEGROUTE = 0x1234`.
- À ce moment, le routeur le transmet à son voisin en utilisant l'adresse IP de destination qui définissait le next-hop. Le compteur de paquets encapsulés est incrémenté. Par la suite, seul le tag présent dans le header `segRoute_t` sera consulté pour forwarder le paquet.
- À chaque saut d'un paquet encapsulé dans un routeur classique, le routeur vérifie si le numéro de point de passage correspond à son numéro de nœud placé dans un registre par le contrôleur.

Si non, il le transmet suivant sa table de routage `"encap_routing"`. Si le routeur est la destination du tunnel, il retire le header `segRoute_t`, remet le type ethernet à `TYPE_IPV4 = 0x800` et les règles de routage IPv4 classiques sont appliquées à ce paquet pour le transmettre.

Les ajouts du TP suivant Traceroutable ont été implémentés afin que des Traceroute en TCP puissent être lancés dans le réseau avec la commande ``traceroute -n -w 0.5 -q 1 -T IP_dst`` et qui produit ce résultat pour la topologie donnée :

```

RAPACE:                                     --- 10.8.2.2 ping statistics ---
Switches:                                  2 packets transmitted, 2 received, 0% packet loss, time 1002ms
  s0: router                               rtt min/avg/max/mdev = 20.455/20.555/20.656/0.100 ms
  s1: router                               root@ir-server:/home/felix.lusseau# traceroute -n -w 0.5 -q 1 -T 10.8.2.2
  s2: router_lw                           traceroute to 10.8.2.2 (10.8.2.2), 30 hops max, 60 byte packets
  s3: router_lw                           1  10.1.1.1   3.241 ms
  s4: router_lw                           2  20.1.2.2   9.561 ms
  s5: router_lw                           3  *
  s6: router_lw                           4  20.6.7.2   99.391 ms
  s7: router                               5  20.7.8.2   146.672 ms
Hosts:                                    6  10.8.2.2   179.138 ms
  h1:                                     root@ir-server:/home/felix.lusseau#

```

Ici, il est possible de voir que chaque routeur a répondu à l'expiration du TTL en mettant comme IP source l'adresse de l'interface d'entrée du paquet. On remarque également que lors de l'encapsulation, le TTL n'est pas décrémenté donc le passage dans un tunnel de longueur variable est masqué sous la forme d'un unique saut.

De plus les routeurs low-energy n'ont pas de Loopbacks ni d'IPv4 sur leurs interfaces (Mininet en génère mais elles ne sont pas utilisées).

Le Traceroute est implémenté par la table "icmp_ingress_port" qui à un port d'entrée associe l'IP associée à cette interface. Cette table est remplie à l'initialisation du contrôleur par la fonction ``set_icmp_ingress_port_table``.

Si un paquet entre dans le routeur avec un TTL égal à 1, le routeur va le drop et renvoyer un paquet ICMP de type ICMP_TTL_EXPIRED = 11 à la source du paquet en mettant son IP en source de cette réponse afin qu'elle puisse apparaître sur le Traceroute.

Cette réponse n'est enclenchée que si le paquet n'est pas encapsulé. Il peut cependant traverser le tunnel avec un TTL à 1 et c'est le routeur de sortie qui renverra la réponse.

Le routeur compte donc les paquets entrants et les paquets qu'il encapsule en étant placé à l'entrée du tunnel.

La seule commande spécifique au routeur est la suivante :

- ``add_encap_node <start_node> <flow> <end_node>``

Cette règle permet à l'utilisateur d'ajouter une règle d'encapsulation entre deux routeurs pour un flux "IP-src IP-dst" donné. Tout le trafic entre ces deux IPs est alors encapsulé dans les deux directions.

4) Router_lw

Ces routeurs Low-Energy sont un code très simplifié par rapport aux routeurs classiques. Ils ne comprennent pas l'IPv4 et ne disposent d'aucune intelligence.

Lors de leur instanciation, leurs contrôleurs calculent les plus courts chemins vers tous les autres switches (pas les hôtes) et les stockent dans une table où l'élément d'entrée est le numéro d'un routeur et les éléments de sortie sont le port de sortie et l'adresse MAC du next_hop vers ce routeur.

S'il reçoit un paquet ne contenant pas l'en-tête `segRoute_t` et le type Ethernet `TYPE_SEGROUTE = 0x1234`, il le droppe car il ne peut rien en faire. Autrement il le forwardé suivant le checkpoint qu'il contient en remplaçant l'en-tête d'encapsulation à l'identique.

Ce routeur n'encapsule pas de nouveaux flux donc il ne compte que les paquets entrants.

Enfin, ces routeurs low-energy ne comprenant que l'encapsulation, ils ne peuvent être placés face à un hôte.

L'ajout des règles empêche de démarrer ou terminer un tunnel sur un routeur low-energy, mais dans les deux cas le paquet serait rejeté car non compréhensible pour le routeur.

Cet équipement n'a pas de commandes spécifiques.

IV – Méta-contrôleur, API et métrologie

- Le méta-contrôleur `meta-controller.py` démarre en premier et avec lui le réseau Mininet. Le lancement du réseau par la commande `sudo python3 meta-controller.py` va d'abord générer le fichier `network.py` qui va décrire le réseau physique sous forme de clique, exporté en `topology.json` par Mininet. L'étape suivante consiste à générer un fichier `logical_topology.json` à partir de celui-ci en retirant les liens inutilisés. Le calcul des routes est effectué depuis `logical_topology.json` par les contrôleurs. Le méta-contrôleur lance ensuite des sous-processus dont les entrées et sorties lui sont pipées après avoir affiché la topologie initiale et avant d'initialiser la CLI en attente de commandes. Le méta-contrôleur interagit avec les contrôleurs en écrivant dans leur stdin et lisant dans leur stdout.
- Chaque switch de la topologie demandée dispose d'un contrôleur qui lui est lié. Il y a 4 contrôleurs différents, un par type d'équipement.
Lorsqu'un contrôleur démarre, il compile et flashe le firmware P4 sur le switch qui lui est associé en utilisant la commande `swap` du fichier `swap.py` qui est une simple mise en fonction et correction des chemins par rapport au script `example_swap.py` fourni avec l'énoncé. Il réinitialise les états et tables et les remplit suivant l'équipement présent. Ensuite les calculs de routes ainsi que la découverte des voisins sont réalisés et la CLI s'active et se met en attente de commandes. Elle n'est pas directement accessible à l'utilisateur mais le méta-contrôleur peut interagir avec (lire et écrire dedans).
- Le code P4 est placé sur le nœud Mininet par le contrôleur qui lui est lié.
Si le contrôleur est stoppé, le dataplane sera figé et continuera à fonctionner dans l'état actuel jusqu'à ce qu'un nouveau contrôleur soit lancé et le réinitialise.

Cette structure répond bien au schéma proposé dans l'énoncé.

La CLI a été implémentée grâce à la bibliothèque Cmd2 de Python.

Elle dispose de toutes les commandes demandées, d'une aide avec la commande `help -v` et de l'auto-complétions ainsi qu'une vérification partielle des arguments.

```
RAPACE_CLI> help -v
Documented commands (use 'help -v' for verbose/'help <topic>' for details):
=====
add_encap_node      <node_src> <flow> <node_dst> - Add an encapsulation node
add_fw_rule         <flow> - Add a firewall rule. A flow is a string of the form 'src_ip dst_ip
                        dst_port protocol'
add_link            <link> - Add a link
add_node            <node_name> <type>
change_weight       <link> <weight> - Change the weight of a link
help               List available commands or provide detailed help for a specific command
history            View, run, edit, save, or clear previously entered commands
quit              Exit this application
remove_link        <link> - Remove a link
see               topology|filters|load|tunnelled|rate - See the topology, the filters, the load,
                        the tunnelled flows or the packet rate
set_port_in        <lb_id> <port_in> - Set the port_in of the loadbalancer
set_rate_lb        <pkts/s> - Set the rate of the loadbalancer
swap              <node_id> <equipment> [args] - Swap the equipment of a node or add one
```

Quelques commandes ont été ajoutées par rapport à l'énoncé dont certaines qui sont intégrées au shell de la bibliothèque.

Liste des commandes de l'API :

- see topology :

Affiche la topologie logique dans le terminal.

- see load :

Affiche pour chaque équipement le nombre de paquets comptés en entrée.

- change_weight <link> <weight> :

Permet de changer le poids d'un lien.

Exemple : change_weight ["s3","s4"] 2 Change le poids du lien entre s3 et s4 à 2.

- add_link <link> :

Ajoute le lien entre deux équipements.

Exemple : add_link ["s3","s4"] Ajoute lien entre s3 et s4.

- remove_link <link> :

Supprime le lien entre deux équipements.

Exemple : remove ["s3","s4"] Supprime le lien entre s3 et s4.

- swap <switch_name> <type> :

Remplace l'équipement présent sur un switch par un autre.

Le meta-controller commence par terminer le processus du contrôleur demandé.

L'équipement continue de tourner quelques instants avec un dataplane figé le temps que le méta-contrôleur instancie un nouveau contrôleur dont la première action sera de flasher le nouveau firmware P4 et de réinitialiser les états.

Exemple : swap s0 load_balancer Instancie un loadbalancer sur s0.

NB: Il y aura un petit souci d'affichage pour la prochaine commande tapée, le tampon de sortie n'ayant pas pu être évacué pour chaque équipement. C'est n'est qu'esthétique.

Difficultés rencontrées :

Nous avons essayé de permettre à l'utilisateur d'ajouter et d'enlever des nœuds. Cependant il semble être extrêmement compliqué, d'ajouter des nœuds (et donc de supprimer sûrement). En effet pour cela il faut utiliser la librairie de Mininet pour pouvoir instancier une interface thrift pour l'api et toutes les informations nécessaires. Or la librairie Mininet est conçue pour instancier un réseau mais pas pour pouvoir ajouter individuellement des nœuds avec l'API, l'ordonnancement et les composants nécessaires. Il aurait fallu donc réécrire des fonctions individuelles tout en comprenant l'ensemble de la librairie. Une autre solution aurait été d'instancier des nœuds supplémentaires au début. Cependant cela irait à l'encontre de l'esprit des réseaux programmables, on ferait tourner des nœuds dans le vide...

V – Test d’une configuration

Reprenons la configuration de la figure 1 en introduction :

```
○ p4@p4:~/RAPACE$ sudo python3 meta_controller.py
```

Ajoutons le port “in” sur le loadbalancer en direction du firewall :

```
RAPACE_CLI> set port in s1 s0
```

Puis connectons-nous sur h1 depuis un nouveau terminal.

```
p4@p4:~/RAPACE$ mx h1
```

Pour récupérer l’IP de h2 regardons la topologie :

```
RAPACE_CLI> see topology
{'Hosts': {'h1': '10.1.1.2/24', 'h2': '10.6.2.2/24'},
 'Links': [['h1', 's0'],
            ['s0', 's1'],
            ['s1', 's2'],
            ['s1', 's3'],
            ['s3', 's4'],
            ['s2', 's4'],
            ['s4', 's5'],
            ['s5', 'h2']],
 'RoutersLoopback': {'s2': '10.100.0.2/32',
                     's3': '10.100.0.3/32',
                     's5': '10.100.0.5/32'},
 'Switches': {'s0': 'firewall',
               's1': 'load_balancer',
               's2': 'router',
               's3': 'router',
               's4': 'router_lw',
               's5': 'router'}}
```

Depuis le second interface on flood ping h2:

```
root@p4:/home/p4/RAPACE# ping -f 10.6.2.2
PING 10.6.2.2 (10.6.2.2) 56(84) bytes of data.
.....
--- 10.6.2.2 ping statistics ---
159 packets transmitted, 4 received, 97,4843% packet loss, time 3266ms
rtt min/avg/max/mdev = 265.840/460.017/737.240/193.699 ms, pipe 36, ipg/ewma 20.670/354.340 ms
```

Le ping est fonctionnel et la limite d’1 paquet par seconde est respectée (4 paquets reçus sur les 4 dernières secondes).

VI – Pistes d’améliorations

Il est possible d’améliorer notre projet en travaillant ces points :

- Meilleure gestion du changement de la topologie. Ici nous jetons toutes les tables à chaque changement. Il est possible de modifier les tables ligne par ligne mais cela devient assez complexe.
- Suppression d’une règle de firewall. Pas explicitement demandé par le sujet mais réalisable facilement. Si l’utilisateur oublie la règle, l’affichage d’une entrée de table proposée par la librairie n’aide vraiment pas à retrouver cette information à moins de décoder de l’hexadécimal dans un format peu compréhensible.
- Ajout et suppression d’un nœud. Aucune piste tangible restante.
- Suppression d’une règle d’encapsulation. Pas explicitement demandé mais réalisable facilement.
- “Partie pour aller plus loin...” tout sauf le routeur low-energy déjà implémenté.

VII – Conclusion

Réponse à la question finale : Que pensez-vous de l’implémentation de tels réseaux malléables en pratique ?

De tels réseaux malléables en pratique possèdent de vrais atouts. Il est possible de pouvoir rapidement adapter son réseau en fonction de la topologie souhaitée, des besoins ou des incidents. C’est un vrai plus vers l’implémentation complète du cloud et de l’intelligence artificielle dans les réseaux. On pourrait ainsi instancier une topologie différente en fonction de la charge du réseau afin de réduire la consommation d’énergie. Ces changements peuvent même être pilotés par une intelligence artificielle capable d’anticiper les charges ou les incidents possibles sur le réseau. L’opérateur ferait ainsi des économies d’énergie et gagnerait en efficacité. Ces implémentations sont par contre complexes à combiner et à implémenter, il faudra attendre une maîtrise de ces technologies pour atteindre un réel gain net d’efficacité.