
# Rapport de projet de réseau programmable

ANDRIANANDRASANA-DINA Vagnona
FILIPOZZI Justin
DEROUSSEAUX Nathanaël

# I - Topologie

Au lancement du programme, la topologie tournant sur le mininet, est bien une clique composé de X switch et X hôtes tel que chaque switch ait un hôte chacun. Le nombre X de switchs est défini par le nombre de switchs voulu dans la topologie logique.
Concernant la topologie logique à l'allumage, elle est définie par un fichier yaml, puis stockée dans un graphe networkX afin d'y calculer les chemins les plus courts.

# II - Implémentation des équipements

**Firewall:** Pour le firewall, nous avons décidé d'utiliser une table "firewall" qui mark_to_drop le paquet dans le cas où son quintuplet (IP source, IP dst, protocole, port source, port destination) correspondrait à un quintuplet ajouter avec la commande add_fw_rule.
Concernant les ports, ne sachant pas à l'avance si il s'agit des ports TCP ou bien des ports UDP nous avons des metadatas 'srcPort' et 'dstPort' qui ont leur valeur attribuée à l'étape "parsing" du paquet suivant si il s'agit d'un paquet UDP ou TCP.

**Load-Balancer:** Pour le load-balancer, si le paquet vient d'un port OUT, alors il est redirigé vers le port IN. Sinon, On calcule la somme de son quintuplet (IP source, IP dst, protocole, port source, port destination) modulo le nombre de port OUT. Ainsi, les paquets venants du port IN sont équitablement répartis parmi les ports OUT, et deux paquets ayant le même quintuplet (IP source, IP dst, protocole, port source, port destination) seront redirigés vers le même port.
Pour ce faire, nous utilisons deux tables, la première permet soit de diriger le paquet vers le port IN si il est originaire d'un des ports OUT, soit d'appeler la deuxième table. La deuxième permet de répartir équitablement les paquets, mais aussi de limiter le débit de paquets que le load-balance transmet.
Malheureusement, bien que nous ayons développé cet équipement, nous n'avons pas eu le temps de l'intégrer au méta-controlleur.

**Routeur:** Le routeur implémenté ici, est un routeur classique qui permet lorsqu'un paquet arrive sur le routeur de l'envoyer par la suite au prochain saut correspondant au chemin le plus court vers la destination calculée au préalable. Il lui manque donc l'encapsulation des paquets qui aurait permis de forcer un point de passage avant que le paquet reprenne son chemin.

# III - Meta-controller, API et métrologie

Le code est divisé en deux parties : une partie CLI, et une partie API.
La partie CLI est une assez basique, et contient uniquement les fonctions pour gérer les entrées utilisateur. Le fichier commands.py est un résumé de tout ce que nous avons implémenté : add_link, change_weight, see_topologie, etc…
Nous n'avons pas implémenté see_tunelled, add_encap_node et aucunes des fonctions relatives au load-balancer.

La partie API est bien plus intéressante :

- __ini__.py contient toutes les fonctions “publiques” de l’api, appelées par le CLI.
- *network.py* est le méta-controlleur : contient la classe représentant l’ensemble du réseau. Ses attributs contiennent la topologie physique (clique mininet), la topologie logique (avec networkx), tous les contrôleurs des différents équipements. Ses différents getters permettent d'accéder à toutes les informations utiles du réseau. Ses fonctions permettent de s’assurer de la cohérence du réseau, elles vérifient que le graphe est connexe, calcule le plus court chemin ou s’assure que les firewall n’ont que deux ports.
- *équipements* contient les classes représentant les équipements. router.py, host.py et firewall.py, sont des classes héritées. Elles permettent chacune d’effectuer les actions propres aux équipements. Par exemple, router.py peut ajouter des règles de routage, ou firewall.py permet de définir les règles de drop du firewall
- utils.py et parser.py contiennent des fonctions utiles et de quoi parser le fichier yaml.

## Que pensons-nous de l’implémentation de tels réseaux malléables en pratique ?

- ☐ Cela est intéressant car il est possible de mettre en place des réseaux très adaptables et efficaces à condition que le méta-contrôleur implémente des fonctions complexes comme l’adaptation automatique intelligente, pour que le réseau se reforme en fonction des besoins et de la charge.