UFR de **mathématique** et d'**informatique**
Université de Strasbourg

Réseaux Programmables
Projet
2023 - 2024
M2-S3

Réseaux Alertes, Programmables et Adaptatifs avec Console d'Edition (RAPACE)

Ce projet est à faire en groupe de 2 ou 3.
Votre projet ainsi que le rapport qui l'accompagne seront à déposer sur Moodle avant le 28 janvier 2024.
Le barême est donné à titre indicatif et peut être modifié suivant l'humeur des correcteurs.

Modalités

Ce projet (en groupe de 2 ou 3) vous demandera de programmer le plan de contrôle (en python), des plans de données (en P4), ainsi qu'une NorthBound Interface (similaire à ce que vous avez manipulé en TP noté).

Votre projet devra fonctionner sur la VM fournie par l'ETH Zurich.

Il vous est demandé de rendre votre code accompagné d'un rapport au format `pdf`. Votre code sera indenté et commenté. Le rapport expliquera ce que vous avez implémenté (ou non) et justifiera vos choix d'implémentation si nécessaire. On rappelle qu'un rapport est différent d'un `README`.

Le projet est à rendre pour le 28 janvier 2024 sur Moodle, dans une archive `.zip`. Le nom de l'archive et le rapport contiendront le nom des membres du groupe.

Votre rapport contiendra votre expérience d'implémentation, les choix effectués, ainsi que les difficultés rencontrées et la réponse à la question posée en fin de sujet. En particulier, vous détaillerez un ou deux scénarios pouvant être effectués par le correcteur pour observer les capacités de votre plateforme au moment du rendu.

Le fond et la forme du rapport (y compris syntaxe, grammaire et orthographe) feront partie intégrante de la notation. La forme du code (indentation, commentaires) sera également prise en compte. Le non-respect de ces consignes entraînera un malus.

Avant de commencer

Ce projet vous demandera de modifier le plan de données des commutateurs P4 à la volée, sans éteindre l'équipement. Cette fonctionnalité est supportée par la cible `simple_switch` et l'API de BMv2, mais certaines modifications doivent être apportées à `p4utils`. Les modifications doivent être effectuées dans `p4utils/utils/thrift_API.py` et `p4utils/mininetlib/node.py`. Les fichiers

modifiés sont fournis sur Moodle. Le nouveau fichier `node.py` rajoute l'option `--enable-swap` à la cible, pour activer la fonctionnalité, tandis que le nouveau fichier `thrift_API.py` corrige un bug mineur de la plateforme. Vous veillerez à récupérer le bon correctif, en fonction de si votre installation est locale ou si vous utilisez la VM de l'ETHZ (dont la version de p4utils est plus ancienne).

Un fichier `example_swap.py` est également fourni pour vous montrer comment implémenter cette fonctionnalité en python dans le contrôleur.

Ce projet vous demandera probablement de consulter la documentation de `p4utils`. Le module `topology` contient notamment des fonctions qui vous seront utiles (en particulier pour extraire certaines informations comme le numéro de port, les adresses MAC, etc). Le git P4learning dispose également de ressources qui pourront vous être utiles.

Vous pouvez de plus consulter la correction du TP noté (mise sur le git du sujet), qui peut servir de base pour votre projet.

Description générale

Le but de ce projet est de concevoir un réseau complètement adaptable et programmable, dont les fonctionnalités se détachent complètement du matériel. Vous concevrez plusieurs data-plane et control-plane, qui seront modifiés à la volée en fonction des requêtes passées par l'utilisateur via une CLI afin de moduler le réseau.

La topologie « physique » (définie dans `network.py` et utilisée par mininet) sera une clique. La topologie logique, contenue dans un graphe networkX, sera celle sur laquelle les plus courts chemins seront calculés. Ainsi, n'importe quelle topologie logique (hors multigraphe) pourra être implémentées.

Dans cette topologie, les commutateurs P4 pourront être utilisés pour implémenter différents types d'équipement (un routeur, un load-balancer, et un firewall).

Le projet sera composé d'un **meta-contrôleur**, qui connaîtra l'état complet du réseau (la topologie logique, ainsi que les types d'équipements implémentés sur chaque commutateur). Il lancera un **contrôleur** par commutateur, en fonction de l'équipement déployé. Les contrôleurs implémentent la logique propre à l'équipement (e.g., le calcul et l'installation des plus courts chemins pour les routeurs). Enfin, une **cli** sera mise à disposition de l'utilisateur. Elle permettra d'afficher des informations sur la topologie ainsi que de modifier cette dernière.

Un exemple de fonctionnement est illustré Figure 1. 8 contrôleurs sont lancés, pour chaque routeur, firewall et load-balancer sur le réseau. L'utilisateur, par la CLI, demande ensuite à changer le load-balancer en routeur. Un nouveau contrôleur est lancé (et le code P4 est également modifié).

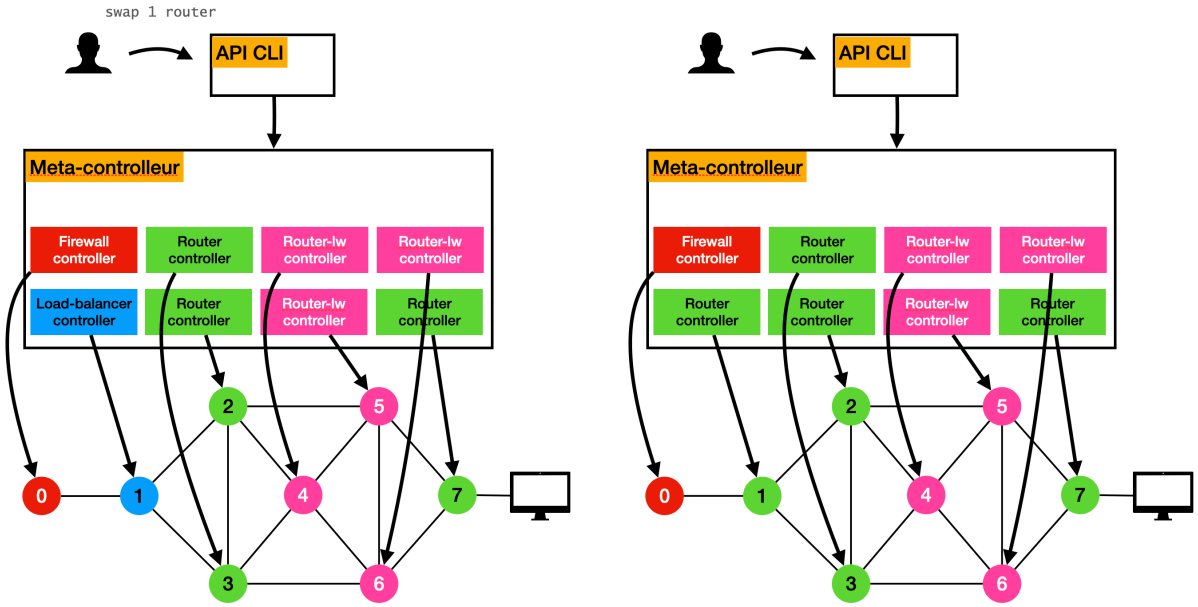

Figure 1 : Illustration de l'architecture du projet

Le reste du sujet ne détaille pas comment implémenter de manière précise chaque équipement. C'est à vous de décider des éventuels choix d'implémentation pouvant survenir. Veillez à justifier ces éventuels choix dans le rapport.

Etape 1 : Topologie (4 points)

Vous commencerez par implémenter la possibilité de démarrer n'importe quelle topologie souhaitée. La topologie souhaitée (initialement) sera fournie par l'utilisateur au lancement et contiendra la liste des noeuds utilisés, l'équipement devant tourner dessus, ainsi que la liste des liens utilisés. Cette topologie logique sera celle utilisée par les équipements pour calculer les chemins. On rappelle que la topologie décrite par mininet sera toujours une clique.

Etape 2 : Implémentation des équipements (10 points)

Vous implémenterez quatre types d'équipement. Vous pouvez, pour aider l'implémentation, vous basez sur les corrigés des exercices faits en TP.

- **Firewall:** Le Firewall ne possède pas de capacité de routage, et possède uniquement deux ports. Le trafic arrivant sur un port est envoyé sur l'autre. Cependant, il est possible pour l'utilisateur de rajouter des filtres, permettant d'empêcher la commutation de certains paquets.

  Plus précisément, L'utilisateur pourra rajouter des règles permettant de *drop* tous les paquets matchant un quintuplet `(IP source, IP dst, protocole, port source, port destination)` via une commande `add_fw_rule <flow>` passée par l'API.

  Le Firewall comptera le nombre de paquets ayant été filtrés par chaque règle ajoutée. Il comptera également le nombre de paquets total reçus.

- **Load-Balancer:** Le Load-Balancer possède un port `in`. Tous ses autres ports actifs sont `out`. Un paquet reçu sur un port `out` est renvoyé sur le port `in`. Un paquet reçu sur le port `in` est envoyé sur un port `out` aléatoire. On notera cependant que le load-balacing sera *flow-aware* : un même flux (identifié par le quintuplet (`IP source, IP dst, protocole, port source, port destination`)) sera envoyé sur le même port out.

  Le Load-Balancer est également responsable de surveiller la charge de trafic envoyée vers chaque équipement. Par défaut, une limite de 1 paquet par seconde sera instaurée sur chaque port. Cette limite peut être modifiée par l'utilisateur via une commande `set_rate_lb <pkt/s>`. On supposera que tous les ports du load-balancer suivent la même *rate-limit*.

  Il comptera également le nombre de paquets total reçus.

- **Router:** : Un routeur classique, comme vu en TP ou lors du TP noté. Cependant, ce routeur aura également la capacité d'encapsuler les paquets. A la manière de Segment Routing, un point de passage pourra être spécifié dans le paquet. Les routeurs en aval routeront le paquet suivant les informations du segment, et non les informations IP. Pour simplifier, on considèrera qu'un seul segment peut être rajouté sur le paquet. Cependant, ce segment peut spécifier un noeud ou un lien (à la manière d'un segment de noeud et d'un segment d'adjacence).

  L'utilisateur peut rajouter une règle d'encapsulation via la CLI. Par exemple, `add_encap_node <flow> <node>` fera rajouter au routeur un segment forçant les paquets du flux `flow` à passer par le noeud `node`.

  Il comptera également le nombre de paquets total reçus ainsi que le nombre de paquets qu'il a encapsulés.

On notera que `p4utils` ne donne pas de loopback aux commutateurs P4 dans le fichier `topology.json`. Vous devez donc les rajouter vous-même sur le graphe networkX utilisé par les commutateurs pour calculer les chemins, et faire en sorte que des chemins sont calculés / installés vers les loopbacks des autres commutateurs. Par défaut (dans le TP `simple_routing`), seuls les chemins vers les hôtes sont calculés.

On conseille également de concevoir votre code de manière que les contrôleurs (et non le meta-contrôleur) soient en charge d'ouvrir une connexion via Thrift sur les commutateurs P4, et d'uploader le data-plane requis. L'instanciation d'un contrôleur démarrera donc par l'ouverture d'une connexion, une réinitialisation des états présents sur le switch, et l'upload du nouveau data-plane compilé.

Etape 3 : Meta-controller, API et métrologie (5 points)

Implémentez le meta-contrôleur et l'API. Le meta-contrôleur sera en charge de connaître l'état actuel de la topologie, et d'opérer les changements demandés par l'utilisateur via l'API. En particulier, votre API offrira la possibilité d'appeler les commandes suivantes (en plus des commandes spécifiques aux équipements évoquées précédemment):

- `swap <node_id> <equipment> [args]` : remplace l'équipement tournant actuellement sur le switch `node_id` par `equipment`. Le control-plane et le data-plane sont modifiés. Il sera possible de rajouter un équipement sur un noeud ne faisant pas partie de la topologie logique. Vous rajouterez à cette fonction les arguments nécessaires au déploiement correct de l'équipement choisi.

- `see topology` : Affiche la topologie logique sous forme d'une liste de liens. L'équipement présentement implémenté sur chaque commutateur est également affiché, ainsi que le poids de chaque lien.
- `change_weight <link> <weight>` : Change le poids du lien spécifié, et re-calcule/réinstalle les plus courts chemins.
- `remove link <link>` : Retire le lien de la topologie logique (i.e., du graphe networkX utilisé pour calculer les chemins).
- `add link <link>` : Ajoute un lien à la topologie logique.

Enfin, on rajoutera la possibilité de consulter les quelques informations métrologiques :
- `see filters` : affiche le nombre de paquets ayant été filtrés par les règles des firewalls
- `see load` : affiche le nombre de paquets reçus par chaque équipement du réseau.
- `see tunnelled` : affiche le nombre de paquets que les routeurs ont encapsulés.

Etape 4 : Pour aller plus loin (Bonus)

Différentes fonctionnalités peuvent être rajoutées (et prises en compte lors de la notation). Notamment :
- **Equipements:** D'autres équipements (au choix) peuvent être implémentés. On peut par exemple envisager l'ajout d'un NAT, ou d'un routeur « Low-Energy », qui n'utiliserait pas de TCAM (et donc pas de *longest prefix match*) et ne comprendrait donc que l'encapsulation.
- **Alertes** : Rajouter des alertes affichées par l'API, par exemple lorsqu'un équipement semble surchargé comparativement aux autres équipements du réseau ou que le *traffic-rate* devient trop important pour certains flux.
- **Gestion intelligente du swap** : Lorsqu'un swap est demandé par l'utilisateur, vérifié que la connectivité peut être maintenue le temps que le swap se produise (e.g., en changeant les poids pour éviter l'équipement pendant le temps du swap). Si c'est impossible, afficher un avertissement à l'utilisateur.
- **Adaptation automatique** : Faites en sorte que le réseau propose (ou implémente lui-même) des solutions aux problèmes détectés (par exemple, en rajoutant un load-balancer ou en dupliquant un équipement en cas de surcharge)

Question (1 point)

Que pensez-vous de l'implémentation de tels réseaux malléables en pratique ?