Rapport du projet Rapace

Thomas Dumond & Ethan Huret

Git: https://github.com/LosKeeper/rapace

Table des matières

I. Introduction

Ce projet vise à concevoir un réseau entièrement adaptable et programmable, indépendant du matériel, où les fonctionnalités peuvent être modifiées dynamiquement en fonction des requêtes de l'utilisateur via une interface en ligne de commande (CLI). La topologie physique du réseau est une clique, utilisée par Mininet, tandis que la topologie logique est représentée par un graphe NetworkX, permettant la mise en œuvre de différents équipements tels que routeurs, load-balancers et firewalls à l'aide de commutateurs P4. Le projet comprend un meta-contrôleur ayant une connaissance complète de l'état du réseau, qui lance des contrôleurs spécifiques pour chaque commutateur en fonction de l'équipement déployé. Ces contrôleurs mettent en œuvre la logique propre à chaque équipement, notamment le calcul et l'installation des plus courts chemins pour les routeurs. Enfin, une interface en ligne de commande est mise à disposition de l'utilisateur, offrant la possibilité d'afficher des informations sur la topologie du réseau et de la modifier selon ses besoins.

Pour lancer le projet, on commencera par lancer le script qui lance la topologie physique dans Mininet:

sudo python phyNet.py --<light|medium|complex> --layer <2|3>

L'option --layer vous permet de choisir la couche du réseau (couche 2 ou 3). L'option --<light|medium|complex> vous permet de sélectionner la topologie du réseau en fonction des topologies prédéfinies dans le dossier topology. La topologie physique est ensuite créée dans le fichier topology.json

Par la suite, on lancera le script permettant la mise en place de la topologie logique en flashant les différents routeurs avec le programme P4 qui leur est associe selon la topologie choisie:

sudo python logiNet.py –input <topologie_yaml>

L'option --input vous permet de choisir le fichier d'entrée du réseau logique parmi les fichiers prédéfinis dans le dossier de topologie. Ainsi la nouvelle topologie logique utilisée par les switches est créée dans le fichier topology_2.json, le réseau est alors en place et l'API est lancé sur lequel on peut interagir avec le réseau par différentes commandes.

Il est aussi possible de définir soi-même la topologie logique. Le format du fichier yaml est le suivant:

```
nodes:
  - name: s1
    type: router
    neighbors: s2 h1
    inflow:
```

- "nodes" renseigne la liste des noeuds
- "name" : est le nom du contrôleur et doit être le même nom qu'un des noeuds de la topologie physique
- "type" : est le type du contrôleur (firewall, load-balancer, router, router-lw)
- neighbors : est la liste des voisins du noeud
- "inflow" : est le noeud du flux entrant pour le load-balancer (ne rien mettre si c'est un autre type de contrôleur)

Il faut faire attention à bien modifier le script phyNet.py si le fichier yaml est modifié. Il faut par exemple augmenter le nombre de switch P4, ajouter des hosts, et faire correspondre les liens entre host et switch P4 car, à contrario des liens entre switchs, les hosts ne sont connectés qu'à un seul switch (les switchs étant tous connectés entre eux).

II. Implémentation

I. Control-plane (python)

Le control plane est chargé de la gestion de la topologie du réseau ainsi que la création et maintenance des tables de routage. Cette partie est codé en python et est organisé selon différentes classes.

I. Méta-contrôleur

La classe MetaController est chargé d'importer la topologie logique sur la topologie physique. Pour cela, cette classe dispose de plusieurs méthodes:
- init_import_logi_topology: est responsable de créer une copie du fichier topology.json afin de garder une sauvegarde du fichier originale délivrée par phyNet.py, et de supprimer les liens et les nœuds non renseignées par l'utilisateur dans le fichier yaml dans la topologie logique (topologie_2.json).
- import_logi_topology: est responsable de créer les contrôleurs renseignés par l'utilisateur dans le fichier yaml (cela prend en compte le type du contrôleur, et utilise la classe correspondante au type), de compiler les fichiers P4 associés ou non selon les arguments rentrés pour logiNet.py, de flasher le fichier P4 sur le nœud de la topologie physique correspondant, et de garder la liste des contrôleurs en mémoire.
- add_loopback: ajoute une adresse IP loopback pour les contrôleurs de type routeur.
- add_node: ajoute un nœud sur la topologie logique uniquement s'il existe dans la topologie physique en se basant sur l'ID du nœud.
- remove_node: supprime un nœud sur la topologie logique.
- add_link: ajoute un lien entre deux nœuds sur la topologie logique seulement s'il existe sur la topologie physique.
- remove_link: supprime un lien entre deux nœuds sur la topologie logique.
- swap_node: change le contrôleur d'un nœud pour un autre type en supprimant l'ancien et en créant le nouveau, puis en reflashant le nœud avec le nouveau contrôleur.
- change_weight: change le poids d'un lien dans la topologie logique.
- set_rate_limit: change la limite de flux pour tous les load-balancers.
- reset_all_tables: réinitialise toutes les tables de tous les contrôleurs.

A noter, que les méthodes opérant des changements sur la topologie logique appellent une méthode update_topologies afin de re-charger la topologie logique sur chaque contrôleur.
Il existe d'autres méthodes qu'uniquement celles présentées ci-dessus, certaines afin de faciliter la programmation et d'autre pour afficher des informations sur les contrôleurs gardés en mémoire (la liste des voisins des contrôleurs, le nombre de paquets comptés par un équipement etc).

II. Contrôleur

La classe Controller est la classe parent de chaque classe-type (Firewall, Load-balancer, Router, RouterLw). Elle implémente les méthodes communes à toutes ces classes:
- compile: compile le fichier P4.
- flash: flashe le fichier P4 compilé sur l'équipement.
- update_topology: met à jour la topologie logique avec celle mise en argument.

Chaque contrôleur a une méthode init_table adaptée selon son type.

III. Firewall

La classe Firewall vise à initialiser les tables du nœud en question en prenant en ajoutant un match sur le port ingress. Ce match est utilisé afin de prendre en compte uniquement les liens de la topologie logique en jetant tous les paquets arrivant sur des liens physiques et non logiques. Cette classe possède une dernière méthode add_rule afin d'ajouter des règles de filtrage directement dans une des tables.

IV. Load-balancer

La classe LoadBalancer initialise principalement les tables du nœud associé. Pour cela, la classe possède la liste des voisins ainsi que le voisin qui fournit le flux entrant. La méthode init_table initialise les tables en ajoutant un match sur le port ingress et associe une action selon que le port soit un port du flux sortant ou le port du flux entrant. Elle initialise aussi une autre table en ajoutant un match sur un nombre compris entre 0 et le nombre de ports du flux sortant et lui associe en argument le port de chaque voisin du flux sortant.

V. Router et Router-Lw

Les classes Router et RouterLw sont les mêmes classes. Ces classes vise à initialiser les tables du nœud associé en ajoutant pour chaque destination possible un match sur l'adresse IP de destination et en ajoutant en argument l'adresse MAC de l'interface du prochain saut (calculé selon le plus court chemin du nœud en question vers la destination sur la topologie logique) sur le lien avec le nœud. L'initialisation des tables comprend aussi l'ajout d'un match sur le port ingress afin d'ajouter l'adresse IP source des paquets arrivant sur le nœud dans l'header ICMP (le but étant de fournir les informations nécessaires au protocole ICMP pour remplir les IP d'un traceroute).

II. Data-plane (p4)

I. Includes

Les fichiers inclus dans tous nos programmes en P4 sont:

- headers.p4: il définit la structure des en-têtes des paquets et des métadonnées utilisées pour tous les différents programmes P4 comme les structures des paquets Ethernet, IPv4, UDP, TCP et ICMP.
- parser.p4: il définit le processus de parsing et de déparsing des paquets en extrayant d'abord l'en-tête Ethernet puis soit IPv4 soit ICMP en fonction de champ type puis en IPv4 UDP ou TCP.
- checksum.p4: il définit les contrôles nécessaires pour la vérification et le calcul des sommes de contrôle.

II. Firewall

Le firewall est composé de 2 tables. La première table entry_port est utilisée pour effectuer une correspondance sur le port ingress. Lorsqu'un paquet entre dans le firewall, le contrôleur Ingress récupère l'action associé au port ingress (si non présent, le paquet est jeté). L'action associé est set_nhop qui remplace le port egress par le port en argument dans la ligne de la table correspondante. Dans le cas du firewall, il n'y a que deux ports possibles vu qu'il possède uniquement deux liens, ainsi si un paquet arrive sur le port 1, il est envoyé sur le port 2 et réciproquement. Ensuite, si l'action set_nhop est effectuée, la table filter_table est appliquée. Cette table matche de manière sur l'adresse IP source et destination, le protocole de transport et le port source et destination. Si toutes les conditions sont remplies, le paquet est jeté et un compteur est incrémenté. Les entrées de cette table sont ajoutées par la méthode add_rule du control-plane.

III. Load Balancer

Le load-balancer est composé de 2 tables. La première table entry_port est utilisée pour effectuer une correspondance sur le port ingress. Lorsqu'un paquet entre dans le load-balancer, le contrôleur Ingress récupère l'action associé au port ingress (si non présent, le paquet est jeté). Si le port est un des ports du flux sortant, l'action associé est set_nhop qui remplace le port egress par le port en argument dans la ligne de la table correspondante. Cependant, si le port est le port du flux entrant,

l'action associée est random_nhop qui calcule le hash du paquet (selon les informations: adresse IP source et destination, protocole de transport, port source et destination). Si cette action est effectuée, la table load_balancer est appliqué. Elle matche sur le hash du paquet. Chaque hash (compris entre 1 et le nombre de ports du flux sortant) est associé à un des ports du flux sortant. Cela permet d'envoyer les paquets de manière aléatoire aux voisins du flux sortant du nœud.

IV. Router

Le routeur est composé de 2 tables. La première table ipv4_lpm est utilisée pour effectuer une correspondance sur le plus long préfixe d'une adresse IP et le port de sortie du routeur. Lorsqu'un paquet IP entre dans le routeur, le contrôleur Ingress va consulter cette table avec l'adresse de destination du paquet et va obtenir en sortie si le routage du paquet est possible un adresse mac ainsi qu'un port de sortie et une action à réaliser:

- Si le prochain saut est vers un host, on applique l'action set_nhop_host qui swap les adresses mac du paquet, positionne le port de sortie, décrémente le TTL et incrémente le compteur de paquets reçus
- Si le prochain saut est vers un router, on applique l'action set_nhop_routeur qui swap les adresses mac du paquet (de manière différente que l'action précédente)n décrémente le TTL in incrémente le compteur du nombre de paquets reçus.
- En l'absence de correspondance, l'action qui sera réalisée sera drop qui va juste jeter le paquet.

Une deuxième table est aussi utilisée dans le cas des paquets ICMP qui est notamment utilisée lors de la réalisation d'un traceroute. Si un paquet IPv4 et TCP arrive avec un TTL inférieur ou égal à 1, alors il revoit un paquet ICMP à l'émetteur grâce à la table icmp_ingress_port qui donne la correspondance entre le port sur lequel le paquet est arrivé et l'IP de ce port. Ainsi, le paquet est renvoyé à l'émetteur et permet alors la réalisation d'un traceroute.

III. API

Après le lancement de l'installation de le topologie logique, l'API est accessible sous forme de CLI. Les commandes implémentées sont:

- swap <nom_du_noeud> <équipement>: Le nœud choisi va être relancé avec l'équipement choisi. Après le lancement de l'équipement sur le nœud, on réinitialise les tables de tous les routeurs pour recalculer le plus court chemin entre chaque nœud.
- see topology: Affiche la topologie actuelle du réseau dans le terminal avec le type d'équipement et les nœuds auquel il est relié. Cependant, pour une meilleure compréhension de la topologie, nous réalisons aussi un graphique de la topologie actuelle dans une image nommée graph.png avec le nom de chaque nœud et en vert les hosts et en rouge les autres équipements.
- change_weight <nœud_1> <nœud_2> <poids>: Ajoute le poids choisi sur le lien entre les 2 nœuds et réinitialise les tables des routeurs pour recalculer tous les plus courts chemins entre les équipements. Ainsi, les paquets risquent de changer de chemins (*Figure 4 - Changement de route après un change_weight*)
- remove_link <nœud_1> <nœud_2>: retire les liens entre les 2 nœuds de la topologie logique et réinitialise les tables des routeurs pour ne plus qu'ils empruntent ce chemin dans leur calcul de plus court chemin.
- add_link <nœud_1> <nœud_2>:ajoute un lien entre les 2 nœuds à la topologie logique (ce lien existe déjà dans le topologie physique) et réinitialise les tables des routeurs pour les autoriser à utiliser ce chemin dans leur calcul de plus court chemin.
- see filters: affiche les paquets bloqués par le ou les firewalls
- see load: affiche les paquets reçus pour chaque équipement

- see tunneled: n'ayant pas réalisé la gestion de l'encapsulation des paquets, nous avons limité la commande à l'affichage des paquets reçus par les routeurs
- add_fw_rule <src_ip> <dst_ip> <udp | tcp> <src_port> <dst_port>: ajoute une règle pour bloquer les paquets matchant sur les entrées (*Figure 3 - Blocage de paquets par le firewall*).
- set_rate_lb <pkt/s>: limite le nombre de paquets par seconde sur le load balancer.

III. Problèmes rencontrés

I. La VM

La plupart des problèmes que nous avons eu étaient dus à la VM QEMU fournie. Nous utilisons tous les deux QEMU sur Windows et programmions directement dessus avec VS Code Server et nous avons eu des problèmes de lenteur à la modification des fichiers mais surtout à la compilation des fichiers P4 ce qui nous a obligé à ajouter des options de lancement a notre programme python qui s'occupe de la compilation des programmes P4 pour soit ne pas compile et réutiliser les programmes déjà compilés existants (avec l'option –no-compile) soit compiler uniquement un fichier (avec l'option –compile <équipement> qui compile uniquement le ficher associé à l'équipement choisi).

II. Router

Un autre problème que nous avons rencontré était lors de l'implémentation du programme P4 des routeurs pour lesquels nous avons récupéré les exemples fournis et n'arrivions pas à faire ping 2 hosts reliés au même routeur. Après différents tests, nous en avons déduit que nous devions avoir des actions différentes au niveau du swap d'adresses mac en fonction de si la sortie est un routeur ou un host (*Figure 5 – Différentes actions du router en fonction du prochain saut*).

IV. Amélioration possible

I. API

Dans notre implémentation, l'API est juste une CLI et pas une vraie API. Ainsi, pour pouvoir lancer les commandes voulues, on doit obligatoirement être sur la même machine ce qui n'est pas optimale. Le mieux aurait été d'implémenter à la place de notre API actuel un serveur avec soit Flask soit Fast API qui serait accessible depuis un client web qui lui pourrait rentrer les commandes voulues en REST. Ainsi on pourrait modifier la topologie ou les équipements sans devoir être connecté forcément au serveur.

II. Router

Nous n'avons pas implémenté l'encapsulation des paquets du type Simple Routing sur les routeurs par manque de temps. Ayants eu quelques problèmes pour déjà réaliser un ping entre 2 hosts, nous avons choisi de nous focaliser sur la résolution de ce problème ainsi que l'implémentation de trace-route.

V. Conclusion

Pour conclure, nous avons implémenté un réseau complètement adaptable et programmables comprenant différents équipements qui est fonctionnel, les routeurs routent correctement les paquets à partir de leur tables (*Figure 1 - Log sur s1 d'un ping de h1 vers h2*, *Figure 2 - Log sur s1 d'un ping de h2 vers h1*), le blocage de paquets du firewall est fonctionnel (*Figure 3 - Blocage de paquets par le firewall*), le changement de poids des liens aussi comme on peut le voir avec la traceroute (*Figure 4 - Changement de route après un change_weight*).

Concernant l'implémentation d'un tel réseau malléable, en pratique, nous ne trouvons pas forcément ça pertinent. Premièrement, c'est assez complexe à mettre en place au vu du nombre d'heures que nous avons passé dessus pour finalement avoir juste des routeur un firewall et un load-balancer alors qu'un tel réseau est bien plus simple et rapide à mettre en place sans utiliser tout ça. De plus, on travaille sur une simulation uniquement mais en réalité, ce genre de matériel coûte cher et est bien plus cher que si nous achetions chaque élément séparément. Même si on peut avoir un réseau vraiment malléable qui réagit exactement comme on souhaite, on ne trouve quand même pas cela pertinent.

VI. Annexe

```
[16:46:38.897] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Processing packet received on port 1
[16:46:38.897] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Parser 'parser': start
[16:46:38.898] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Parser 'parser' entering state 'start'
[16:46:38.898] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Extracting header 'ethernet'
[16:46:38.898] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Parser state 'start': key is 0800
[16:46:38.900] [bmv2] [T] [thread 5901] [14.0] [cxt 0] Bytes parsed: 14
[16:46:38.900] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Parser 'parser' entering state 'parse_ipv4'
[16:46:38.900] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Extracting header 'ipv4'
[16:46:38.900] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Parser state 'parse_ipv4' has no switch, going to default next state
[16:46:38.900] [bmv2] [T] [thread 5901] [14.0] [cxt 0] Bytes parsed: 34
[16:46:38.900] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Parser 'parser': end
[16:46:38.900] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Pipeline 'ingress': start
[16:46:38.900] [bmv2] [T] [thread 5901] [14.0] [cxt 0] /home/p4/p4-tools/rapace/p4src/router.p4(58) Condition "hdr.ipv4.isValid()" (node_2) is true
[16:46:38.900] [bmv2] [T] [thread 5901] [14.0] [cxt 0] Applying table 'RIngress.ipv4_exact'
[16:46:38.900] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Looking up key:
* hdr.ipv4.dstAddr    : 0a000002

[16:46:38.900] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Table 'RIngress.ipv4_exact': hit with handle 1
[16:46:38.900] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Dumping entry 1
Match key:
* hdr.ipv4.dstAddr    : EXACT     0a000002
Action entry: RIngress.set_nhop - a000002,2,

[16:46:38.900] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Action entry is RIngress.set_nhop - a000002,2,
[16:46:38.900] [bmv2] [T] [thread 5901] [14.0] [cxt 0] Action RIngress.set_nhop
[16:46:38.900] [bmv2] [T] [thread 5901] [14.0] [cxt 0] /home/p4/p4-tools/rapace/p4src/router.p4(31) Primitive hdr.ethernet.srcAddr = hdr.ethernet.dstAddr
[16:46:38.900] [bmv2] [T] [thread 5901] [14.0] [cxt 0] /home/p4/p4-tools/rapace/p4src/router.p4(34) Primitive hdr.ethernet.dstAddr = dstAddr
[16:46:38.900] [bmv2] [T] [thread 5901] [14.0] [cxt 0] /home/p4/p4-tools/rapace/p4src/router.p4(36) Primitive standard_metadata.ingress_port = standard_met
[16:46:38.900] [bmv2] [T] [thread 5901] [14.0] [cxt 0] /home/p4/p4-tools/rapace/p4src/router.p4(39) Primitive standard_metadata.egress_spec = port
[16:46:38.900] [bmv2] [T] [thread 5901] [14.0] [cxt 0] /home/p4/p4-tools/rapace/p4src/router.p4(42) Primitive hdr.ipv4.ttl = hdr.ipv4.ttl - 1
[16:46:38.900] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Pipeline 'ingress': end
[16:46:38.900] [bmv2] [D] [thread 5901] [14.0] [cxt 0] Egress port is 2
[16:46:38.910] [bmv2] [D] [thread 5904] [14.0] [cxt 0] Pipeline 'egress': start
[16:46:38.910] [bmv2] [D] [thread 5904] [14.0] [cxt 0] Pipeline 'egress': end
[16:46:38.910] [bmv2] [D] [thread 5904] [14.0] [cxt 0] Deparser 'deparser': start
[16:46:38.910] [bmv2] [D] [thread 5904] [14.0] [cxt 0] Updating checksum 'cksum'
[16:46:38.911] [bmv2] [D] [thread 5904] [14.0] [cxt 0] Deparsing header 'ethernet'
[16:46:38.911] [bmv2] [D] [thread 5904] [14.0] [cxt 0] Deparsing header 'ipv4'
[16:46:38.911] [bmv2] [D] [thread 5904] [14.0] [cxt 0] Deparser 'deparser': end
[16:46:38.911] [bmv2] [D] [thread 5906] [14.0] [cxt 0] Transmitting packet of size 98 out of port 2
```

FIGURE 1 - LOG SUR S1 D'UN PING DE H1 VERS H2

```
[16:46:42.900] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Processing packet received on port 2
[16:46:42.900] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Parser 'parser': start
[16:46:42.900] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Parser 'parser' entering state 'start'
[16:46:42.900] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Extracting header 'ethernet'
[16:46:42.900] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Parser state 'start': key is 0800
[16:46:42.900] [bmv2] [T] [thread 5901] [15.0] [cxt 0] Bytes parsed: 14
[16:46:42.900] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Parser 'parser' entering state 'parse_ipv4'
[16:46:42.900] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Extracting header 'ipv4'
[16:46:42.900] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Parser state 'parse_ipv4' has no switch, going to default next state
[16:46:42.900] [bmv2] [T] [thread 5901] [15.0] [cxt 0] Bytes parsed: 34
[16:46:42.900] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Parser 'parser': end
[16:46:42.900] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Pipeline 'ingress': start
[16:46:42.900] [bmv2] [T] [thread 5901] [15.0] [cxt 0] /home/p4/p4-tools/rapace/p4src/router.p4(58) Condition "hdr.ipv4.isValid()" (node_2) is true
[16:46:42.900] [bmv2] [T] [thread 5901] [15.0] [cxt 0] Applying table 'RIngress.ipv4_exact'
[16:46:42.900] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Looking up key:
* hdr.ipv4.dstAddr    : 0a000001

[16:46:42.900] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Table 'RIngress.ipv4_exact': hit with handle 0
[16:46:42.910] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Dumping entry 0
Match key:
* hdr.ipv4.dstAddr    : EXACT     0a000001
Action entry: RIngress.set_nhop - a000001,1,

[16:46:42.910] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Action entry is RIngress.set_nhop - a000001,1,
[16:46:42.910] [bmv2] [T] [thread 5901] [15.0] [cxt 0] Action RIngress.set_nhop
[16:46:42.910] [bmv2] [T] [thread 5901] [15.0] [cxt 0] /home/p4/p4-tools/rapace/p4src/router.p4(31) Primitive hdr.ethernet.srcAddr = hdr.ethernet.dstAddr
[16:46:42.910] [bmv2] [T] [thread 5901] [15.0] [cxt 0] /home/p4/p4-tools/rapace/p4src/router.p4(34) Primitive hdr.ethernet.dstAddr = dstAddr
[16:46:42.910] [bmv2] [T] [thread 5901] [15.0] [cxt 0] /home/p4/p4-tools/rapace/p4src/router.p4(36) Primitive standard_metadata.ingress_port = standard_met
[16:46:42.910] [bmv2] [T] [thread 5901] [15.0] [cxt 0] /home/p4/p4-tools/rapace/p4src/router.p4(39) Primitive standard_metadata.egress_spec = port
[16:46:42.910] [bmv2] [T] [thread 5901] [15.0] [cxt 0] /home/p4/p4-tools/rapace/p4src/router.p4(42) Primitive hdr.ipv4.ttl = hdr.ipv4.ttl - 1
[16:46:42.910] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Pipeline 'ingress': end
[16:46:42.910] [bmv2] [D] [thread 5901] [15.0] [cxt 0] Egress port is 1
[16:46:42.910] [bmv2] [D] [thread 5903] [15.0] [cxt 0] Pipeline 'egress': start
[16:46:42.910] [bmv2] [D] [thread 5903] [15.0] [cxt 0] Pipeline 'egress': end
[16:46:42.910] [bmv2] [D] [thread 5903] [15.0] [cxt 0] Deparser 'deparser': start
[16:46:42.910] [bmv2] [D] [thread 5903] [15.0] [cxt 0] Updating checksum 'cksum'
[16:46:42.910] [bmv2] [D] [thread 5903] [15.0] [cxt 0] Deparsing header 'ethernet'
[16:46:42.910] [bmv2] [D] [thread 5903] [15.0] [cxt 0] Deparsing header 'ipv4'
[16:46:42.910] [bmv2] [D] [thread 5903] [15.0] [cxt 0] Deparser 'deparser': end
[16:46:42.910] [bmv2] [D] [thread 5906] [15.0] [cxt 0] Transmitting packet of size 98 out of port 1
```

FIGURE 2 - LOG SUR S1 D'UN PING DE H2 VERS H1

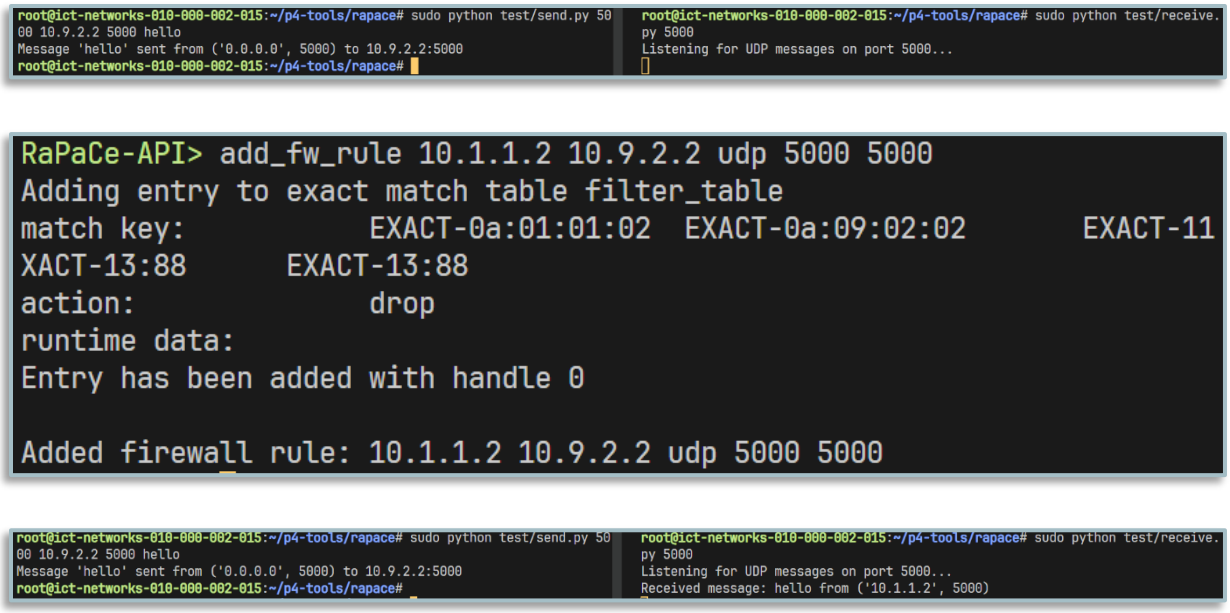

FIGURE 3 - BLOCAGE DE PAQUETS PAR LE FIREWALL

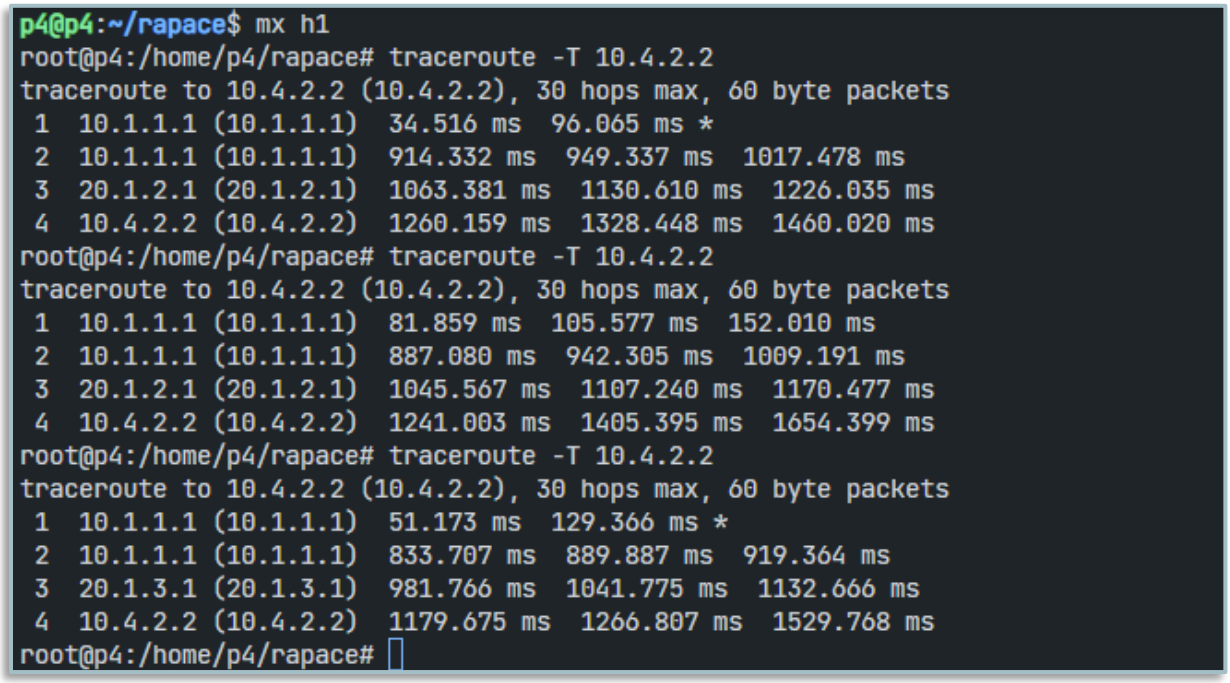

FIGURE 4 - CHANGEMENT DE ROUTE APRES UN CHANGE_WEIGHT

```
action set_nhop_host(macAddr_t dstAddr, egressSpec_t port) {
    /* swap mac addresses for the packet to go back to the host */
    // set the destination mac address with the source mac address from the packet
    hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
    // set the source mac address with the one we get from the table
    hdr.ethernet.srcAddr = dstAddr;
    // set the output port that we also get from the table
    standard_metadata.egress_spec = port;
    // decrease ttl by 1
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;

    // count
    total_packets.read(tmp, 0);
    total_packets.write(0, tmp + 1);
}

action set_nhop_router(macAddr_t dstAddr, egressSpec_t port) {
    // set the source mac address with the previous destination mac adddress
    hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    // set the destination mac address with the one we get from the table
    hdr.ethernet.dstAddr = dstAddr;
    // set the output port that we also get from the table
    standard_metadata.egress_spec = port;
    // decrease ttl by 1
    hdr.ipv4.ttl = hdr.ipv4.ttl - 1;

    // count
    total_packets.read(tmp, 0);
    total_packets.write(0, tmp + 1);
}
```

FIGURE 5 – DIFFERENTES ACTIONS DU ROUTER EN FONCTION DU PROCHAIN SAUT