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

Réseaux Programmable
TP noté
2023 - 2024
M2-S1

Détection de pannes grises en P4 : Heuristic to Expose Link Problems (HELP)

Ce TP est une épreuve individuelle. Les documents sont autorisés.
Le barème est donné à titre indicatif et peut être modifié suivant l'humeur du correcteur
Vous veillerez à déposer vos fichiers .p4, .py et texte sur Moodle dans le temps imparti.

Ce sujet contient 4 pages

Durée : 2h

Présentation

Les pannes grises sont des pannes indétectables par le plan de contrôle. La plupart du temps, tous les paquets ne sont pas affectés, empêchant le plan de contrôle de détecter l'erreur.

Le meilleur moyen de détecter des pannes grises est de vérifier que les paquets émis sont bien reçus. Une différence de compte permet d'inférer la présence d'une panne grise sur le lien traversé.

Dans ce TP noté, vous implémenterez un mécanisme de détection de pannes grises simplifié. Vous devrez compléter un code à trou, à la fois en P4 et en Python. Des questions plus théoriques sont proposées en bonus.

La base du code est celle utilisée pour l'exercice *simple_routing* vu en TP. Vos commutateurs sont donc en mesure d'appliquer les décisions de routage du contrôleur.

Fonctionnement général

Le code fourni est commenté et peut-être compris en l'état. Cette section a pour but de faciliter votre compréhension en expliquant le mécanisme HELP dont vous devez compléter l'implémentation.

HELP fonctionne en comparant le nombre de paquets reçus au nombre de paquets envoyés sur un même lien. Ces nombres sont envoyés au contrôleur, qui en déduit la fiabilité du lien. Le poids d'un lien qui n'est pas fiable est augmenté d'une valeur `DEBUFF` afin de l'exclure des plus courts chemins, qui sont ensuite recalculés.

Lors de l'envoi un paquet, le switch émetteur incrémente le nombre de paquets envoyés sur le port en question. Après `THRESHOLD` paquets envoyés, le switch émetteur envoie le nombre de paquets émis à son voisin dans un header dédié `cnt` rajouté au paquet et réinitialise le compteur du port.

À la réception d'un paquet, le switch récepteur incrémente le nombre de paquets reçus sur ce port. À la réception d'un message contenant un header `cnt`, un switch envoie au contrôleur un digest contenant le nombre contenu dans le header (i.e., le nombre de paquets émis par son voisin), accompagné du compte de paquets reçus sur ce port et du numéro de port sur lequel le message a été reçu.

Le contrôleur utilise les digests pour détecter si le lien est fiable et modifier son poids en conséquence.

Pour éviter un overhead trop important, seule une certaine proportion des paquets émis sont comptés. Cette probabilité peut varier en fonction des ports. Si un paquet a été choisir pour être compté en émission, un tag est ajouté dans le champ `flags` du header IP, pour que le switch voisin le compte également en réception. Les paquets ne contenant pas le tag ne sont pas comptés. Les liens switch - hôte ne sont pas monitorés.

Les compteurs sont gérés à l'aide de registres.

La topologie du réseau est décrite dans la Figure 1.

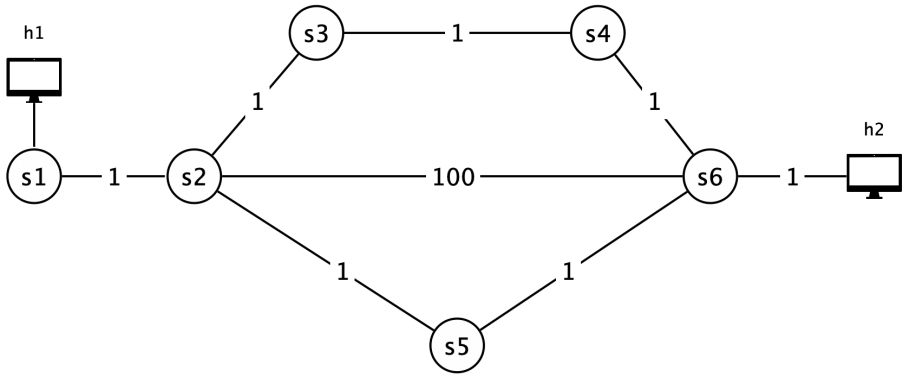

Figure 1: Topologie du réseau

La plupart des fonctionnalités nécessaires sont déjà implémentées. Le code doit uniquement être complété par endroit, tel que défini par les TODOs suivants.

Pour vous aider

Le réseau doit être démarré avec la commande `sudo python network.py`. Le contrôleur doit être lancé avec la commande `sudo python routing-controller.py` une fois que le réseau a démarré. Le contrôleur installe les routes dans les commutateurs.

- `routing-controller` vous donne accès à une CLI qui vous permet d'afficher l'état du réseau à l'aide de la commande `show_links_state`. Vous pouvez également y indiquer qu'un lien est de nouveau stable pour lui faire retrouver son poids initial via `repair s1-s2`. La CLI vous indique aussi les changements éventuels de poids. Une commande `help` est disponible.
- Vous pouvez effectuer des pings comme dans les autres TP, via `h1 ping h2`. Vous pouvez observer le TTL pour vérifier le chemin parcouru.

- Vous pouvez utiliser `send.py` pour générer des pertes. Par exemple, `h1 send.py --n-pkt 5 --n-drops 2 --fail-hops 3` va simuler la perte au troisième saut de 2 des 5 paquets envoyés. Pour des raisons de lisibilité, il est recommandé de limiter le nombre de paquets envoyés à `THRESHOLD`.

TODOs 1 : Parser, Deparser (5 points)

Implémentez le parser, qui parsera un header Ethernet, suivi d'un header CNT (à déclarer) si ce-dernier est présent, avant de parser le header IPv4 puis TCP (pour simplifier, nous considérerons uniquement le protocole TCP à la couche 4).

Les headers d'Ethernet, d'IPv4 et de TCP sont déjà fournis. Les constantes nécessaires sont également définies en amont du fichier `headers.p4`.

Les emplacements à compléter sont marqués de la balise `TODO1` dans le code, dans les fichiers `parser.p4` et `headers.p4`.

TODOs 2: Logique de l'Ingress (5 points)

La logique de l'Ingress consiste à gérer la réception des messages `CNT`, émettre les digests et compter les paquets taggés en réception.

Lorsqu'un paquet est reçu, on rappelle que le commutateur doit incrémenter le compte des paquets reçus sur le port si (et seulement si) le paquet est taggé. Ensuite, si un header `CNT` est également présent, un digest contenant le compteur des paquets en réception, en émission (indiqué dans le header) et le numéro de port de réception doit être envoyé au contrôleur, avant de commuter le paquet comme un paquet IPv4 standard et de reset le compteur. Dans les autres cas, le paquet doit simplement être commuté.

TODOs 3: Logique de l'Egress (6 points)

La logique de l'Egress consiste à décider si un paquet sortant doit être compté suivant une certaine probabilité (si le voisin est un switch), et partager le compteur avec le voisin en aval quand un nombre suffisant de paquets (défini par `THRESHOLD`) ont été comptés. On notera que le header `CNT` ne doit pas être rajouté si le paquet est envoyé en direction d'un hôte. On rappelle également qu'un paquet compté par l'Ingress doit être taggé pour le voisin en aval.

On supposera l'existence d'une table `check_port`, contenant pour chaque port *switch-à-switch* la probabilité qu'un paquet soit compté et fournissant ce paramètre à l'action `check_if_count`. Les ports host-switch ne sont pas présents dans la table. Vous devez programmer l'action `check_if_count`, qui met la variable `count_packet` à `true` si le paquet doit être compté. On rappelle l'existence de la function `random(out_value, hi lo)`, mettant une valeur comprise entre `hi` et `lo` dans `out_value`.

TODOs 4: Contrôleur (4 points)

Le contrôleur contient de nombreuses fonctions, mais la plupart d'entre elles n'ont pas besoin d'être consultées.

Vous devez cependant

1) Compléter la fonction `fill_check_port_table` afin de compléter la table `check_port` utilisée par l'Egress. Une entrée sera rajoutée pour chaque port switch-à-switch afin d'appeler l'action `check_if_count` avec une probabilité de 50% de compter les paquet.
2) La gestion du digest dans `check_port` . Les valeurs des deux compteurs et du port doivent être extraits et ajouté à la liste de triplets renvoyée par la fonction. Les digest parsés sont ensuite utilisés par le contrôleur pour définir si les liens sont fiables (code fourni).
3) La modification du poids des liens suivant la valeur des digests. La fiabilité du lien sera vérifiée en s'assurant que le lien est fiable dans les deux sens. Si ce n'est pas le cas, son poids sera augmenté, et les chemins re-calculés.

Questions bonus (2 points)

1) En pratique, les pertes réseaux peuvent impacter le trafic de contrôle comme les données. Étudiez le comportement de votre solution si le(s) paquet(s) contenant le compteur venai(en)t à se perdre. La méthode implémentée peut-elle encore être utile ? Argumentez.
2) Donnez au moins deux limites ou améliorations potentielles à la méthode implémentée. Argumentez.