Télécom
ParisTech

Télécom
SudParis

Mémoire de stage

Sécurité d'un contrôleur SDN : ONOS

Julien Schoumacher

Diplôme préparé : Ingénieur
Stage effectué du 20 juillet 2016 au 20 janvier 2017 à
Télécom SudParis sous la direction de Grégory Blanc

Remerciements

Avant d'entamer la lecture de ce rapport, je tiens avant tout à remercier toute l'équipe du département RST (Réseaux et Services des Télécommunications) de Télécom SudParis qui m'a si bien accueilli durant ce stage. Je remercie également mon encadrant côté Télécom ParisTech Rida Khatoun, m'ayant mis en relation avec le maître de conférences Gregory Blanc qui m'a encadré avec bienveillance pendant toute la durée du stage. Enfin, toutes les autres personnes que j'ai pu cottoyer plus ou moins longtemps à l'occasion d'évènements ponctuels comme la conférence RAID qui s'est tenue en septembre.

Table des matières

1 Réseau SDN (Software Defined Network) 3
- 1.1 Motivation . . . 3
- 1.2 Concepts . . . 5
- 1.3 Historique . . . 7
- 1.4 Exemples d'applications . . . 9

2 Cadre de l'étude 12
- 2.1 Problématique et objectifs . . . 12
- 2.2 Architecture d'un réseau SDN . . . 13
  - 2.2.1 Openflow . . . 13
  - 2.2.2 Contrôleur SDN . . . 17
  - 2.2.3 ONOS . . . 19
- 2.3 Surface d'attaque . . . 21
  - 2.3.1 Menaces au niveau de l'interaction avec les switchs 21
  - 2.3.2 Menaces au niveau de l'interaction utilisateur . . 21
  - 2.3.3 Autres menaces . . . 21
- 2.4 Scénarios envisagés . . . 21

3 Audit 21
- 3.1 Man in the middle au niveau de l'interface sud . . . 21
- 3.2 Altération de la topologie depuis l'interface sud . . . 21
- 3.3 Deni de service au niveau de l'interface sud . . . 21
- 3.4 Deni de service au niveau de l'interface nord . . . 21
- 3.5 Fuites d'information au niveau de l'interface nord . . . 21
- 3.6 Mauvaise configuration au niveau de l'interface nord . . 21

4 Validation et évaluation 21
- 4.1 Résultats de l'étude . . . 21
- 4.2 Autres considérations . . . 21

5 Perspectives à l'issue du stage 21

6 Ressources 21

1 Réseau SDN (Software Defined Network)

1.1 Motivation

Bien qu'il soit préférable d'éviter les approches rasoirs lorsqu'on souhaite introduire un sujet quelconque, on ne peut pas dans le cas de cette étude portant en partie sur les réseaux SDN, oublier de mentionner quelques éléments difficilement contestables sur les réseaux actuels[1] :

- ※ La demande ne cesse de croître : on observe un accroissement considérable des enjeux liés au traitement de masse importante de données, de l'utilisation de services cloud, du trafic mobile et peut être bientôt de l'utilisation d'objets connectés. Or tous ces éléments présentent le point commun de communiquer avec de nombreuses entités situées sur des réseaux potentiellement éloignés. Cela mobilise donc un trafic réseau intense.
- ※ Les technologies actuelles pour soutenir cette demande énorme sont capables de fournir un débit titanesque : que l'on considère des technologies sans fil ou non, au coeur des réseaux tant au niveau des terminaux des utilisateurs, on atteint aujourd'hui des débits théorique de l'ordre du Gigabit par seconde pour l'utilisateur. Tout cela sans que l'on ait vraiment conscience des conditions que cela requiert.
- ※ Les méthodes d'accès sont aujourd'hui bien différentes. Précédemment le modèle client/serveur était largement employé, avec dans le cas d'une entreprise, un réseau interne constitué de plusieurs LAN séparés, et connecté à internet de manière quasiment unique. Cela entrainant une configuration possiblement statique et donc aisée, le trafic se déroulant principalement sur un mode requête/réponse. Or la tendance, notamment à cause des deux premiers points, est à l'émergence de nouveaux modes d'accès plus horizontaux. Ce type de communication tient entre autres de la distribution plus éparse des données à travers le réseau : grossissement de la taille des bases de données, duplication de celles-ci (mise en cache sur différents serveur à travers le monde pour permettre un accès plus rapide),

1. Aussi résumé dans `https://www.opennetworking.org/sdn-resources/sdn-definition`

augmentation du trafic volumineux (vidéo, voix) et de nouveaux trafics (IoT (Bring Your Own Device), ...) même au sein de l'entreprise. Enfin, l'utilisation de plus en plus répandue de services cloud, avec ses implications en terme de virtualisation (que ce soit des applications, ou bien des bases de données), susceptible de changer en permanence la localisation des serveurs pour garantir une certaine flexibilité.

Or, le réseau principal global tel que nous le connaissons (la partie reposant sur TCP/IP en tout cas) a été conçu d'abord dans un but de résilience : chaque paquet doit être reçu, et peu importe la route empruntée. L'architecture distribuée actuelle n'est donc pas batie pour assurer spécifiquement extensibilité, ni qualité de service définie. Le routeur (et le réseau d'ailleurs) des années 1980 a donc été progressivement amélioré sur la base de ce paradigme initial, avec le plan de données et le plan de contrôle attachés aux mêmes équipements, configurés en partie manuellement. Tout s'est complexifié également : nouveaux protocoles, ajouts d'équipements capables de répartir la charge réseau, filtrer les paquets, prévenir de certaines tentatives d'attaque, etc ...
Certains éléments de réflexion peuvent éventuellement nous mettre sur la voie d'une complexité qui, à défaut d'être exponentielle, l'est d'avantage que simplement linéaire (c'est du moins une conviction personnelle non vérifiée) :

- Plus il y a d'éléments statiques dans un réseau, et plus la modification en profondeur de celui-ci est coûteuse (puisqu'il faut penser à chaque impact sur les parties statiques).
- Si un problème survient, dans le même ordre d'idée, il est difficile d'avoir une vue globale de ce qui se passe puisqu'aucune vue globale du réseau n'est accessible : il faut vérifier (potentiellement) que chaque élément se comporte correctement et est bien configuré.
- Les interfaces entre switchs, routeurs et autres éléments peuvent varier selon le constructeur, et le logiciel sur les équipements est souvent propriétaire et complexe.

De nombreux problèmes se résolvent avec un niveau d'abstraction supplémentaire[2]. Si le développement de systèmes de plus en plus complexes s'est fait de manière très rapide sur PC, c'est d'abord grâce à la première couche d'abstraction qu'ont concstituées les instructions assembleur, puis à la seconde qu'a été le système d'exploitation. Certaines personnes ont eu l'idée, au lieu de considérer le réseau comme un élément périphérique, de le voir comme un processeur capable d'exécuter des instructions basiques, fournissant de fait un service plus facilement adaptable. C'est sur ce principe que repose le Software Defined Network (SDN). Avec une couche d'abstraction supplémentaire que constitue le protocole choisi pour véhiculer le flot d'instructions (Openflow dans notre cas, mais il y en a d'autres que nous évoquerons succintement plus tard), et un système d'exploitation spécifique (Network Operating System, NOS), l'idée est de découpler les différents chemins qu'empruntent les données et le plan de contrôle, à la manière d'un système d'exploitation qui sépare le code d'un programme et les données qu'il utilise.

1.2 Concepts

On peut poursuivre la dernière analogie (avec un ordinateur) de la manière suivante. Sur un PC classique, on crée et utilise des applications qui reposent sur un système d'exploitation responsable des éléments matériels. C'est la même chose dans un modèle SDN : on souhaite créer des applications "réseau" sans se soucier de la manière dont les éléments de ce dernier vont s'organiser pour répondre à la problématique. On sépare ainsi la couche applicative, le système d'exploitation réseau et la communication entre les entités du réseau.

Pour réaliser cela, il est nécessaire, puisqu'un réseau est constitué d'entités physiquement séparées, de disposer d'un protocole de communication standard entre celles-ci. Mais ça n'est pas suffisant : si tous les éléments communiquent entre eux, encore faut-il qu'ils aient des

2. https://en.wikipedia.org/wiki/Fundamental_theorem_of_software_engineering

choses à se dire pour faire circuler, outre les données à faire transiter, les instructions qui vont leur indiquer quoi faire avec ces données. Cela n'est possible que si il existe un cerveau central qui coordonne les opérations (il n'existe pas vraiment d'intelligence collective à ce jour). C'est le rôle du contrôleur SDN. On déporte ainsi l'intelligence humaine déployée dans la configuration de tous les éléments du réseau vers un seul (même si il peut être dupliqué).

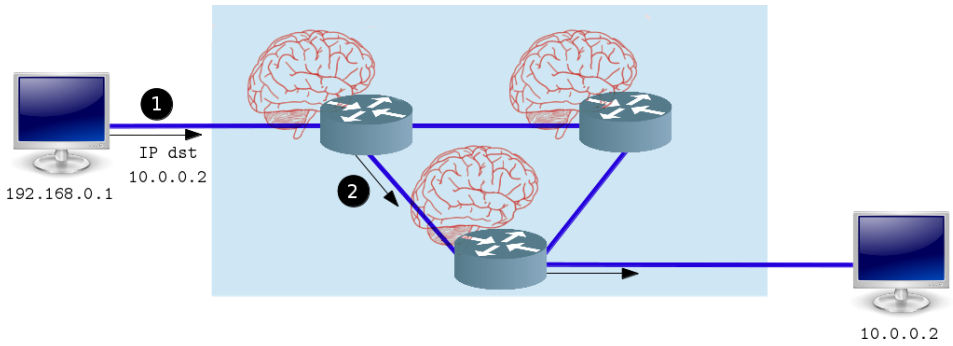

FIGURE 1 – Réseau classique : chaque routeur contient une partie de la logique de contrôle

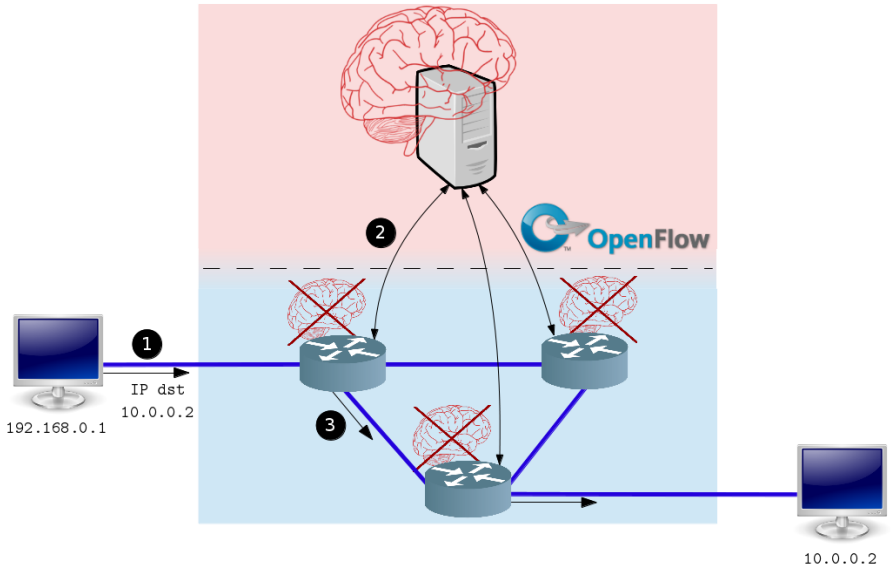

FIGURE 2 – Réseau SDN : "l'intelligence" est déportée vers le contôleur [3]

Les avantages de cette architecture sont multiples :

※ D'abord cela réduit grandement la complexité de configuration manuelle et le risque d'erreur (si le système d'exploitation réseau est

3. Schéma extrait du mémoire "Étude d'OpenFlow dans le contexte de la sécurité" de Maxence Tury

fiable).

- Cela facilite donc énormément le développement d'applications réseau complexes, puisque la tâche peut être quasiment séparée de sa réalisation physique.
- Les routes optimales sont plus facilement calculables qu'au sein d'un réseau classique : un seul élément gère les différentes distances et métriques qui peuvent changer selon le trafic, et être modifiées à la volée par des applications spécifiques.
- La gestion du réseau devient plus simple, les évènements importants (perte d'un lien, disfonctionnement, ralentissement ...) peuvent être remarqués rapidement, la réaction pouvant être automatique et quasiment instantanée.
- Les coûts matériels sont diminués puisque seuls subsistent les switchs "de base" et le contrôleur qui n'est pas réellement un équipement spécifique. Le reste peut être pris en charge logiciellement au niveau du contrôleur.

Bref, pour résumer, beaucoup plus de flexibilité est permise par cette approche, économisant temps et matériel. Evidemment, l'idée n'est pas nouvelle, mais n'a pas que des avantages. Notamment : la sécurité du contrôleur devient un point brûlant, puisque toute la gestion du réseau provient de lui.

1.3 Historique

La ressource "A Survey of Software-De ned Networking : Past, Present, and Future of Programmable Networks" [4] présente un état de l'art à la date du 19 janvier 2014 (donc assez récemment pour avoir une vue globale de l'évolution de ce type de technologie). En voici un rapide résumé complété :

Assez tôt lors de l'essor d'internet tel que nous le connaissons, des idées pour fournir une sorte d'API réseau ont émergées (milieu des années 1990 environ) : le groupe Open Signaling propose un protocole

4. https://hal.inria.fr/hal-00825087/file/hal_final.pdf

d'accès universel au matériel (switchs) permettant de distribuer facilement des nouveaux services, pendant qu'Active Networking propose un mécanisme de propagation de code que l'équipement réseau exécute lorsqu'il reçoit les paquets encapsulant le code (ce qui pose au passage un énorme problème de sécurité).

Dans le même temps, le groupe DCAN (Devolved Control of ATM Networks) propose une approche qui se rapproche très fortement du paradigme SDN : convaincus que les fonctions de contrôle et de gestion des différents éléments du réseau doivent être séparées du routage des données et déléguées à des équipements spécialisés, ils développent un protocole minimaliste entre contrôleur et autres équipements, à la manière du protocole Openflow aujourd'hui majoritaire dans les réseaux SDN.

Le projet 4D [5] initié en 2004 (et semblant s'être fini un peu avant 2010), ajoute quant à lui des abstractions diverses (découverte des voisins proches et remontée des informations, dissémination d'informations sur l'état général du réseau, puis prise de décision sur la base des informations récoltées). C'est ce genre de projet qui a inspiré l'idée de système d'exploitation réseau, qu'implémentent les contrôleurs SDN.

On peut encore citer NETCONF et Ethane (2006), le premier pouvant être vu comme une extension de SNMP, le second comme un ancêtre immédiat d'openflow (un contrôleur qui décide si et où un paquet devrait être redirigé, et des switchs qui constitués d'une table de flux et d'un canal sécurisé vers le contrôleur).

Openflow a quant à lui précédé l'apparition du terme SDN lors d'expérimentations à Stanford vers 2010 (la première spécification d'Openflow pour la production (1.0.0), a été publiée début 2010).

5. http://www.cs.cmu.edu/~4D/

1.4 Exemples d'applications

En 2011, l'Open Networking Foundation (ONF) est créée. Regroupant des gros acteurs comme Google, Yahoo, Facebook, Verizon, Microsoft ou encore Deutsche Telekom, c'est l'organisme principal qui encourage l'adoption de la technologie SDN, en publiant régulièrement de nouvelles spécifications Openflow.

Google, en 2012, présente, pour la première fois, une architecture SDN pour ses datacenters, utilisant Openflow sur des switchs conçus par eux-mêmes (étant pionniers, leur position étant qu'ils auraient utilisé des switchs existants si ceux-ci implémentaient toutes les fonctionnalités Openflow leur étant nécessaires). Grâce à ce nouveau paradigme, Google affirme (en 2012) obtenir des performances dix fois supérieures en terme de débit, et surtout utiliser 100% [6] de leurs lignes (contrairement aux 30 à 40% en vigueur dans l'industrie, notamment pour garantir un service même en cas de nombreuses pannes, ce qui n'est plus nécessaire avec un réseau SDN qui adapte automatiquement le routage pour pallier aux problèmes).

Microsoft semble également s'être intéressé au SDN pour son service "Azure" depuis quelques années maintenant [7], et de manière générale toutes les figures de proue de l'industrie numérique (celles qui disposent de nombreux serveurs et gèrent des flux énormes et grandissants de données comme Amazon, AT&T, Facebook, ...) se sont plus ou moins annoncées investies dans le processus, possiblement avec leur propre protocole SDN. Si des grosses entreprises ont annoncé l'utilisation de ce type de réseau, les données précises concernant les performances obtenues sont difficilement accessibles, ce qui rend compliquée l'évaluation des avantages réels de SDN.

Par ailleurs, les switchs compatibles Openflow, ou les switchs Openflow seuls, que fournissent (depuis 2011 environ) certaines entreprises majeures du domaine comme Brocade, HP, IBM, Juniper ou encore

6. http://www.networkworld.com/article/2189197/lan-wan/google-s-software-defined-openflow-backbone-drives-wan-links-
html
7. http://www.networkworld.com/article/2937396/cloud-computing/microsoft-needs-sdn-for-azure-cloud.
html

NEC, Pronto ou Pica8, manquent encore parfois de maturité (RFC parfois interprétée différemment, vulnérabilités spécifiques, ...). L'évolution des versions Openflow est également parfois difficile à suivre (la spécification de la version 1.0.0 (fin 2009) fait 42 pages [8], celle de la dernière version stable (1.5.0, fin 2014) en fait 277 [9]). Or l'industrie dans ce domaine présente une certaine inertie (peu sont les compagnie qui proposent des switchs compatibles avec la dernière version d'Openflow, certaines vendant encore des switchs Openflow 1.0).

Les applications semblent donc d'un premier abord limitées aux datacenters (beaucoup de données à traiter, grande variabilité de la topologie (les machines virtuelles changent souvent d'emplacement/adresse)). En réalité, SDN peut être utilisé de manière effective dans plusieurs cas :

- Réseaux d'entreprises/universités : dans le cas de nombreux équipements/protocoles différents utilisés, SDN permet théoriquement de faciliter le déploiement de politiques réseau complexes.
- Réseaux optiques : faciliter la transition, la gestion et l'incorporation des réseaux optiques au sein du réseau actuel.
- Infrastructure based WAN (réservé aux entreprises disposant de nombreux serveurs à travers le monde) : à la manière de Google, un moyen de connecter de grosses entités en évitant les goulots d'étranglement. Egalement un moyen envisageable pour un utilisateur de se connecter depuis n'importe quel endroit en disposant des services auxquels il souscrit.
- Infrastructure personnelle, petites entreprises : surveiller le trafic et alerter l'administrateur local ou le fournisseur internet en cas de détection d'activité réseau suspecte (utile notamment dans le cadre de l'IoT).

Au final, une technologie pouvant sembler prometteuse, mais demeurant assez peu utilisée pour plusieurs raisons. Nous allons en étudier

8. http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf
9. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf

deux d'entre elles par la suite : la sécurité générale du réseau, et l'importance capitale du contrôleur qui devient quasiment l'unique clé de voûte du système.

2 Cadre de l'étude

2.1 Problématique et objectifs

Mon stage, dont le sujet n'était pas complètement fixé au départ, a pris la tournure suivante : d'abord une étude des réseaux SDN (ne connaissant pas le domaine), puis début d'expérimentations sur le protocole Openflow, avec Scapy et Wireshark. Ensuite, constitution d'un rapide état de l'art en matière d'attaques (générales) sur les réseaux SDN. Puis, l'orientation s'est faite sur l'étude plus précise d'un contrôleur SDN particulier (ONOS), avec la conception de scénarios d'attaque, suivie de leur réalisation.
Comme on l'a dit au-dessus, le contrôleur SDN est le point névralgique de toute l'infrastructure. Si on le compromet d'une manière où d'une autre, les conséquences peuvent être désastreuses. L'étude a donc eu pour but de déterminer les principaux vecteurs d'attaque envisageables dans ce genre de réseau, principalement concernant le contrôleur ONOS (principalement, parce qu'il est possible d'appliquer une majorité des attaques sur d'autres contrôleurs, même si cela n'a pas été expérimentalement vérifié).
Pour résumer, un audit (se voulant le plus exhaustif possible, même si il est impossible de couvrir l'ensemble des vulnérabilités d'un tel élément logiciel) a été réalisé. Cet audit a été conçu informellement à partir de la méthode STRIDE[10] (utilisée par microsoft à la base, cette manière de modéliser les menaces dans le domaine de la sécurité s'est beaucoup répandue). Certains scénarios ont été expérimentés pour prouver la faisabilité d'attaques précises (donc sous certaines hypothèses qui sont décrites). D'autres faiblesses sont également détaillées dans leur aspect théorique sur la base de sources externes. Bien que n'ayant pas eu accès à une situation réelle de déploiement SDN, j'essaierai de donner une conclusion pas trop biaisée aux tests effectués, et formulerai quelques recommandations et remarques.

10. Spoofing, Tampering, Repudiation, Info disclosure, Denial of service, privilege Escalation

2.2 Architecture d'un réseau SDN

Avant toute chose, il est nécessaire de détailler la façon dont un réseau SDN fonctionne, afin que les attaques qui seront évoquées plus tard puissent être bien comprises. Pour cela, on insistera particulièrement sur la description du protocole Openflow, mais aussi sur celle de l'objet de notre étude, ONOS.

2.2.1 Openflow

Openflow est un des protocoles qui permet de séparer le plan de données et le plan de contrôle (protocole majoritairement utilisé à l'heure actuelle, étant non propriétaire et porté par l'ONF). Pour l'utiliser, il est nécessaire de disposer de switchs compatibles, c'est à dire des switchs capables de gérer des paquets Openflow. Ainsi, les décisions prises au niveau du contrôleur sont transmises aux switchs concernés par le biais de ce protocole. Pour que l'abstraction de la gestion du réseau soit intéressante, le protocole permet aux switchs une gestion assez fine des paquets reçus à leur niveau, et ce jusqu'au niveau 4 (pour TCP comme on va le voir après). C'est le principal élément de ce qu'on appelle l'interface sud d'un réseau SDN (interface entre contrôleur et entités matérielles) :

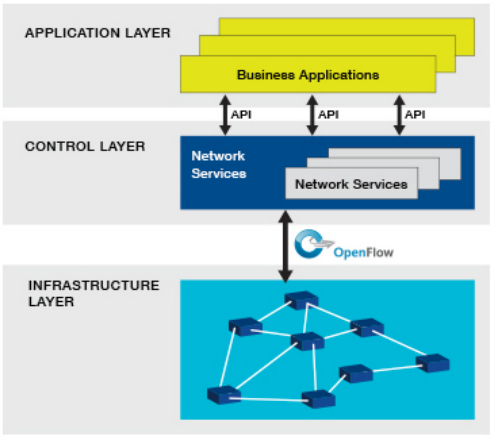

FIGURE 3 – Différentes interfaces, Openflow = interface sud

Chaque switch (qu'il soit compatible Openflow ou virtuel) consiste en plusieurs tables de flux et une table de groupe (non présente en version 1.0), ainsi qu'un (ou plusieurs) canal(aux) vers le(s) contrôleur(s)

(sécurisé(s) via TLS obligatoire en version 1.0 mais obligation supprimée dès la version 1.1 pour des raisons de facilité de déploiement). La description du fonctionnement qui suit est celle de la version 1.3 du protocole (ça n'est donc ni la dernière version, ni la première, mais celle que j'ai principalement utilisée durant le stage, Wireshark la disséquant complètement).

La notion de flux est essentielle. Un flux est constitué de 3 parties :

- La première une règle qui filtre les paquets : en fonction des attributs du paquet à l'entrée du switch (port d'entrée physique, adresse ethernet source, adresse ethernet destination, type de paquet, VLAN id, VLAN priority, adresse IP source, adresse IP destination, protocole IP, ToS IP, TCP port source, TCP port destination entre autres). Il est possible de générer des filtrages généraux avec l'utilisation de jokers (wildcard en anglais) sur les champs souhaités (par exemple, si on veut autoriser toutes les adresses ethernet source ayant pour adresse IP x.x.x.x, on pourra utiliser un joker sur le champ adresse ethernet source).
- La seconde est un compteur qui permet de tenir à jour, si le switch le permet, des statistiques sur l'utilisation du flux.
- La troisième est une action à appliquer en cas de correspondance du paquet : si le paquet remplit les conditions du filtre, plusieurs types de traitements sont possibles. Entre autres : envoyer le paquet au contrôleur, rediriger le paquet vers un port physique spécifique, vers une table de flux, vers tous les ports sauf le port d'entrée, vers les switchs voisins mis à jour par spanning tree, supprimer le paquet, ou encore modifier certains champs avant redirection, rediriger le paquet vers une queue. Bref, toutes les opérations envisageables sur un paquet. Certaines de ces actions doivent être prises en charge par les switchs Openflow pour que ceux-ci puissent être considérés comme tels. D'autres actions sont optionnelles (par exemple la redirection vers les switchs mis à jour par spanning tree).

Chaque nouveau flux ajouté au switch par le contrôleur l'est dans une table de flux spécifiée. Ainsi, plutôt que d'avoir à organiser re-

lativement la priorité de chaque flux, des regroupements peuvent se faire par table pour factoriser certains traitements. Le switch parcourt chaque table de flux (en considérant le premier flux de chaque table) jusqu'à trouver un filtrage correct pour le paquet. A ce moment, le paquet parcourt et subit les traitements de chacun des flux dans la table trouvée (c'est la notion de pipeline Openflow) jusqu'à ce qu'une action de redirection soit trouvée. La redirection peut également être faite vers une table de flux de priorité inférieure (pour éviter qu'un paquet boucle indéfiniment).

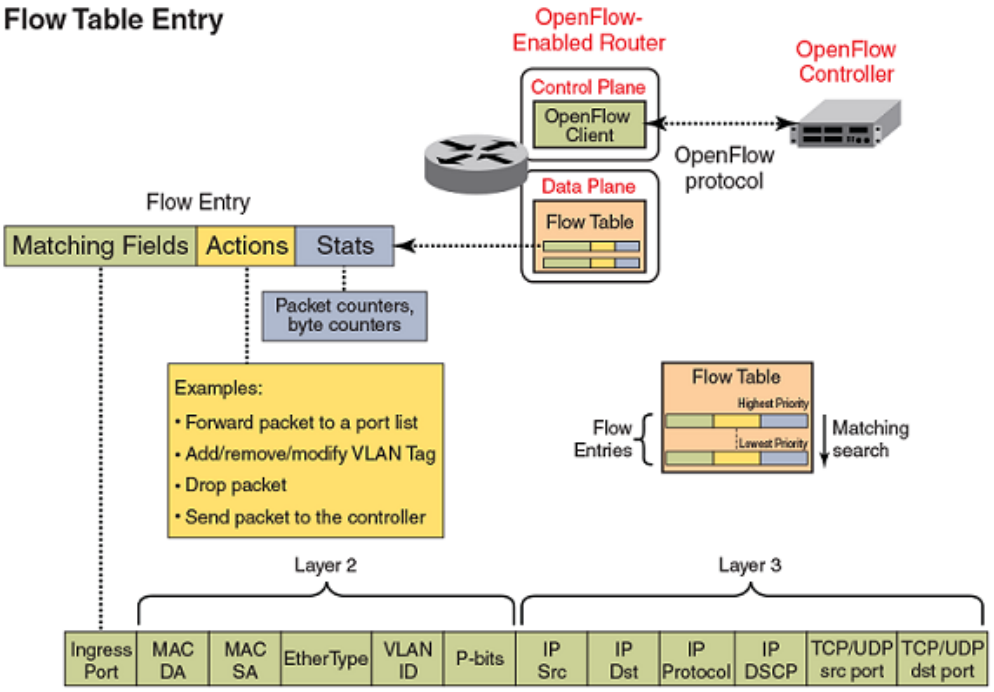

FIGURE 4 – Flux et table de flux

Basiquement, on peut résumer le protocole Openflow comme étant le protocole permettant :

- de mettre à jour ces tables de flux : ajouts, ajouts partiels (par exemple ajouts de précisions à un flux avec joker), suppressions, modifications, etc ....
- d'obtenir des statistiques sur certains éléments (en Openflow 1.3, on peut accéder aux statistiques des flux, des tables, des ports, des queues, des groupes, ou encore du débit (pour la qualité de service)).

- d'obtenir des informations sur les entités du réseau (nom de l'équipement, nom du fabricant, débit supporté ...)

C'est un protocole qui s'établit de manière classique sur une session TCP (éventuellement surmonté d'une session TLS), habituellement sur les ports 6633 ou 6653 (ce dernier étant maintenant alloué pour cet usage par l'IANA).
Il est constitué :

- de messages symétriques (Hello, Echo), qui s'utilisent pour ou démarrer une session Openflow ou s'assurer que les deux parties sont encore connectées
- de messages contrôleur vers switch (ressemblant à du requête/réponse), généralement les messages qui permettent au contrôleur d'obtenir des informations sur le switchs et de lui donner des ordres. Mais aussi, et c'est important, un message qui permet directement d'insérer du trafic dans le réseau. Ainsi, lorsque le switch envoie un paquet qu'il ne sait pas traiter au contrôleur, celui-ci peut le renvoyer au switch (après l'avoir traité et éventuellement modifié), ce qui évite de perdre le paquet [11].
- de messages asynchrones émis par les switchs, qui sont par exemple susceptibles d'informer le contrôleur lorsqu'ils ont modifié, supprimé ou ajouté un flux, mais également lorsqu'un paquet ne correspond à aucun flux, qu'une erreur survient ou bien qu'un port physique change de configuration. Le message asynchrone le plus important est sûrement celui qui survient lorsque le switch doit envoyer le paquet au contrôleur [12] (soit parce qu'une règle le demande explicitement, soit parce que le switch ne peut pas traiter le paquet et que la règle par défaut associée demande un envoi au contrôleur).

La figure précédente provient de mes expérimentations, j'ai en effet été amené durant mon stage à créer une sorte de mini-switch virtuel très basique avec Scapy, ce qui m'a forcé à implémenter à la fois la pile TCP et la session Openflow . Même si par la suite cela m'a été peu

11. On appelle alors ce paquet OFPT_PACKET_OUT (selon la spécifiation) qu'on abrégera par la suite en PACKET_OUT
12. Paquet appelé OFPT_PACKET_IN (selon la spécifiation) qu'on abrégera par la suite en PACKET_IN

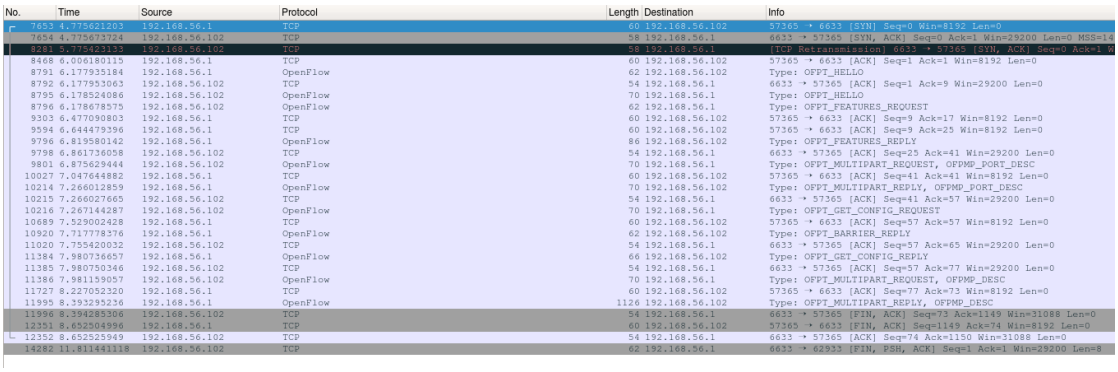

FIGURE 5 – Capture d'une session Openflow réalisée sous Wireshark entre le switch virtuel crée en 192.168.56.1 et le contrôleur en 192.168.56.102

utile pour réaliser mes scénarios d'attaque, cela m'a permis de bien comprendre le protocole.

Le lecteur désirant rentrer dans les détails techniques peut se référer à la spécification de la version 1.3 [13].

2.2.2 Contrôleur SDN

Le contrôleur est, comme on l'a déjà dit précédemment, l'élément central du réseau SDN, puisqu'il offre au niveau de son interface nord, une API pour développer des applications réseau, et, au niveau de son interface sud, il contrôle les entités réseaux se chargeant du plan de données, avec le protocole Openflow.

Pour fonctionner correctement, le contrôleur doit avoir la représentation interne la plus exacte possible de la topologie réseau qu'il dirige. Pour cela, Openflow prévoit certains paquets spécialisés. Mais ce n'est pas suffisant, puisque les switchs eux-mêmes ne sont pas capables de renseigner le contrôleur sur la topologie alentours. C'est pourquoi certains mécanismes sont mis en place (qui dépendent généralement du contrôleur, même si, devant utiliser des protocoles classiques compréhensibles par des switchs, les possibilités restent limitées).
Le mécanisme que j'ai été amené à constater est celui de l'utilisation de paquets LLDP fabriqués par le contrôleur et envoyés aux switchs sous forme de PACKET_OUT. En recevant un tel paquet, un switch va le retransmettre en broadcast aux switchs alentours, qui, normale-

13. https ://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf (106 pages, dont 49 d'explications)

ment, sont configurés pour le renvoyer en PACKET_IN au contrôleur (comportement par défaut, si aucun flux gérant ce type de paquet n'est spécifié, ce qui est préférable). Or, un PACKET_IN encapsule toutes les informations nécessaires au contrôleur pour mettre à jour la topologie locale : en vérifiant que c'est bien lui qui est à l'origine de l'émission du paquet LLDP initial (avec un champ spécial par exemple), il sait que le switch émetteur du PACKET_IN est relié au switch auquel il avait précédemment envoyé un PACKET_OUT, ces premiers étant des encapsulations de paquets réels circulant sur le réseau, on peut donc y lire des adresses ethernet, des adresses IP, ... La question de la confiance relative à la réception de tels paquets est cruciale et on va voir par la suite qu'il est relativement aisé d'attaquer le contrôleur par ce biais.

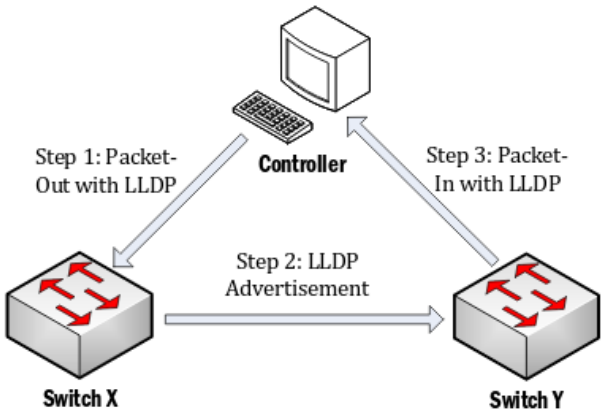

FIGURE 6 – Mécanisme de découverte de topologie par envoi de paquets LLDP

Au sein du contrôleur même, on trouve tout le logiciel nécessaire pour lier les informations reçues depuis les différentes entités du réseau aux intentions de plus haut niveau émises. Comme sur un système d'exploitation classique, on peut trouver une abstraction plus ou moins riche : selon la maturité du contrôleur et le dynamisme de la communauté qui le porte, on trouvera ainsi de nombreuses différences dans la quantité de développement à fournir pour arriver à un même résultat. Par exemple sur ONOS il est possible de spécifier uniquement une intention de haut niveau qui sera automatiquement traduites en règles qui seront si c'est possible envoyées aux switchs.

Le contrôleur offre finalement une API plus ou moins fournie qui permet aux utilisateurs d'écrire des applications en disposant d'abstractions susceptibles de lui éviter l'écriture de code fastidieux.

Parmi les contrôleurs SDN les plus connus, on trouve notamment NOX (premier contrôleur SDN, 2008, rendu open source depuis), POX (juin 2011, en python), OpenDayLight (avril 2013, open source, crée par The Linux Foundation), ONOS (l'objet final de ce stage, décembre 2014, open source, développement repris par The Linux Foundation en 2015), et bien d'autres (Beacon, RoseMary, Ryu ...). Certains projets se ressemblent énormément au niveau des choix effectués (langages utilisés, paradigmes ...). Par exemple OpenDayLight et ONOS sont deux contrôleurs très proches dans ce qu'ils offrent (java, architecture OSGi, sécurité vue comme un élément crucial, ...).

2.2.3 ONOS

ONOS[14] est un contrôleur SDN open source récent (début en décembre 2014, la fondation Linux arrive en octobre 2015 dans le projet), écrit en java, déployable avec Maven et utilisant apache-karaf comme conteneur OSGi (qui fournit entre autre l'interface utilisateur permettant l'interaction avec le contrôleur). La version actuelle est Goldeneye (juin 2016) (1.6), la prochaine Hummingbird. C'est un projet basé sur la technique[15] :

> goal is to « provide an environment that thrives on technical meritocracy. Merit is based on technical contribution, not on financial contribution. »

Le contrôleur est architecturé de la manière suivante :

- Au Sud : une API (southbound) gérant plusieurs protocoles (dont Openflow (toutes les versions jusqu'à la version 1.5, la version 1.6 étant en cours de prise en charge)). C'est la partie d'ONOS qui se charge de la communication avec les switchs. Il est ainsi possible de prendre en charge de nouveaux protocoles ou de nouveaux drivers.

14. Open Network Operating System
15. `http://onosproject.org/governance/`

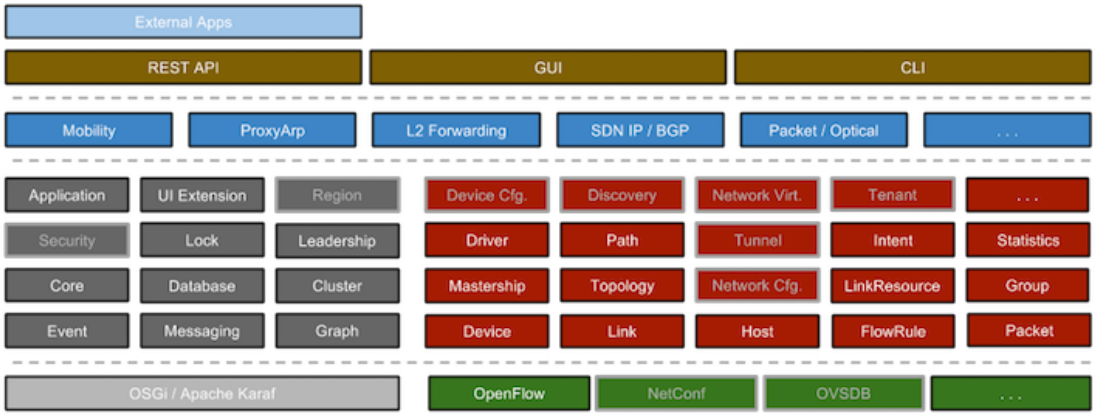

FIGURE 7 – Architecture logicielle d'ONOS (schéma extrait du site officiel)

- Sur le côtés : un protocole d'échange entre contrôleurs. Cela permet un contrôle partagé du réseau. Pour cela, des informations sur la topologie de celui-ci doivent être échangées entre contrôleurs.
- Au Nord : une API (northbound) permettant d'écrire des applications utilisant les ressources offertes par le contrôleur. Si le secure mode est activé (nous reviendrons plus en détail sur cela ultérieurement), l'ensemble des méthodes utilisables est restreint.
- Au Nord encore : une interface utilisateur fournie par Karaf et composée de 3 parties : une API REST accessible sur le port 8181 (configurable), une interface web (permettant de visualiser l'état du réseau, les applications lancées ...) accessible sur ce même port, et une CLI accessible en SSH sur le port 8101 (ou bien directement sur le contrôleur, là encore tout est configurable).

2.3 Surface d'attaque

2.3.1 Menaces au niveau de l'interaction avec les switchs

2.3.2 Menaces au niveau de l'interaction utilisateur

2.3.3 Autres menaces

2.4 Scénarios envisagés

3 Audit

3.1 Man in the middle au niveau de l'interface sud

3.2 Altération de la topologie depuis l'interface sud

3.3 Deni de service au niveau de l'interface sud

3.4 Deni de service au niveau de l'interface nord

3.5 Fuites d'information au niveau de l'interface nord

3.6 Mauvaise configuration au niveau de l'interface nord

4 Validation et évaluation

4.1 Résultats de l'étude

4.2 Autres considérations

5 Perspectives à l'issue du stage

6 Ressources