

Network Data Analytics & Blockchain

Encadrant : M. Potop-Butucaru, G. Bu

**Etudiants : N. Bouchama, Y. Fodil,
M. Guedrez, R. Sekhri**

Table des matières

| | | |
|----------|---|-----------|
| 1 | Cahier des charges | 3 |
| 1.1 | Contexte du projet | 3 |
| 1.2 | Objectifs | 3 |
| 1.3 | Choix techniques | 3 |
| 2 | Plan de développement | 4 |
| 3 | Bibliographie | 6 |
| 4 | Analyse | 7 |
| 4.1 | Objectif | 7 |
| 4.2 | Spécification des besoins fonctionnels | 7 |
| 4.3 | Démonstration | 8 |
| 4.4 | Contraintes | 9 |
| 5 | Conception | 10 |
| 5.1 | Développement | 10 |
| 5.2 | Implémentation d'un routage niveau 3 et d'un monitoring sur le réseau | 11 |
| 5.3 | Les protocoles indépendants à implémenter | 18 |
| 6 | Compte rendu | 20 |
| 6.1 | Approche prise | 20 |
| 6.2 | Monitoring et stockage de données | 21 |

Liste des figures

| | |
|--|----|
| Figure 1 Diagramme de GANTT | 6 |
| Figure 2 Démonstration des applications utilisées | 8 |
| Figure 3 Méthode de prog d'un protocole indé pour le traitement de paquets | 10 |
| Figure 4 Structure générale de code P4 | 10 |
| Figure 5 Description de la topologie en .json | 11 |
| Figure 6 Vue globale de la topologie | 11 |
| Figure 7 Structure des en-têtes standards | 12 |
| Figure 8 Structure de l'en-tête personnalisé | 12 |
| Figure 9 Analyse des en-têtes en entrée du switch | 13 |
| Figure 10 Routage ECMP | 13 |
| Figure 11 Remplir l'en-tête personnalisé avec les champs de monitoring | 14 |
| Figure 12a Création d'une socket de connexion | 14 |
| Figure 12b Monitoring et stockage de données | 15 |
| Figure 13 Prompt mininet | 16 |
| Figure 14 Phase de test | 16 |
| Figure 15 Résultat du monitoring | 17 |
| Figure 16 Stockage des clés-valeurs | 18 |
| Figure 17 En-tête du segment modifié | 19 |

1 Cahier des charges

1.1 Contexte du projet

Dans le cadre de notre projet de la première année en master informatique parcours réseau à l'université Pierre et Marie Curie, nous avons choisi la mise en œuvre d'une architecture Blockchain dans le but de récupérer les données relatives au trafic réseaux, et ce à une granularité suffisante.

1.2 Objectifs

L'objectif de notre projet est d'étudier la faisabilité d'utiliser les architectures blockchain afin de récupérer les données relatives au trafic réseaux, et ce à une granularité suffisante, via le déploiement d'une blockchain à travers le réseau de switches.

Cette alliance de télémétrie et de blockchain repose sur deux approches complémentaires, le marquage et le comptage des flots de données via des méthodes de hachage de type min-sketch et l'enregistrement de cet historique sur une blockchain construites de manière distribué par les switches eux même.

1.3 Choix techniques

C'est les choix préliminaires qui ont été fait en se basant sur les objectifs fixés au départ.

1.3.1 Outils

- **Ubuntu** : Ubuntu est un système d'exploitation GNU/Linux open source libre, gratuit, sécurisé et convivial basé sur la distribution LinuxDebian.¹
- **Vagrant** : Vagrant est un logiciel libre et open-source pour la création et la configuration des environnements de développement virtuel. Il peut être considéré comme un wrapper autour de logiciels de virtualisation comme VirtualBox.²
- **VirtualBox** : est un outil de virtualisation de poste de travail créé par la société Oracle. Il est utilisé pour la mise en place d'un environnement de test ou de développement en exécutant plusieurs systèmes d'exploitation en tant que machines virtuelles (VM) sur un seul PC hôte que ce soit Linux ou Windows.³
- **Wireshark** : est un analyseur de paquets libre et gratuit. Il est utilisé dans le dépannage et l'analyse de réseaux informatiques, le développement de protocoles, l'éducation et la rétro-ingénierie.⁴

¹ <https://ubuntu.com/>

² <https://www.vagrantup.com/>

³ <https://www.virtualbox.org/>

⁴ <https://fr.wikipedia.org/wiki/Wireshark>

- **Mininet** : est un logiciel open source qui est utilisé pour simuler des Software defined network (SDN). Ce logiciel utilise un système de contrôleurs, de switchs et d'hôtes. Mininet supporte le protocole OpenFlow.⁵
- **Sublime Text** : est un éditeur de texte générique codé en C++ et Python, disponible sur Windows, Mac et Linux.⁶

1.3.2 Langages de développement

- **Python** : est un langage de programmation de haut niveau interprété pour la programmation générale. Il a une philosophie de conception qui met l'accent sur la lisibilité du code, notamment en utilisant des espaces importants. Il fournit des constructions qui permettent une programmation claire à petite et à grande échelle.⁷
- **JSON** : JavaScript Object Notation : un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de présenter l'information structurée comme le permet XML par exemple.⁸
- **P4** : est un langage de programmation pour contrôler les plans de données des paquets dans les périphériques réseau, tels que les routeurs et les commutateurs. Contrairement à un langage à usage général tel que C ou Python, P4 est un langage spécifique au domaine avec un certain nombre de constructions optimisées pour la transmission de données réseau. P4 est distribué sous forme de code open source, sous licence permissive, et est géré par « P4 Language Consortium » une organisation à but non lucratif.⁹

2 Plan de développement

Premièrement, nous avons consacré le 1er mois à la recherche et à la documentation, où nous avons commencé par analyser deux documents fournis par notre encadrant portant sur les projets Netchain et DisBlockNet, en parallèle de recherches bibliographiques sur les SDN et la Blockchain. En outre, après la bonne compréhension des documents cités précédemment et l'établissement d'un lien entre les deux, nous nous sommes ensuite penchés sur le plateforme sur laquelle se basera notre projet. Le choix s'est porté sur le duo P4-Mininet.

Nous avons décidé avec notre encadrant d'utiliser un langage P4-16 pour le développement des architectures réseau dont les protocoles de traitement de données sont indépendant, nous nous sommes d'abord renseignés sur la création du langage, nous avons passé en revue le cours d'un master ETH Zurich qui traite du sujet de création de protocoles indépendants sur ce langage.

On s'est ensuite hâté à créer la machine virtuelle avec les composants essentiels cités

⁵ <http://mininet.org/>

⁶ https://fr.wikipedia.org/wiki/Sublime_Text

⁷ <https://docs.python.org/fr/3.5/tutorial/>

⁸ https://fr.wikipedia.org/wiki/JavaScript_Object_Notation

⁹ [https://en.wikipedia.org/wiki/P4_\(programming_language\)](https://en.wikipedia.org/wiki/P4_(programming_language))

dans le cahier de charges. Une fois la machine créée, nous avons testé les quelques protocoles de base fournis mis en place par l'équipe en charge du projet P4-16.

Une fois cette étape confirmée avec l'encadrant, il nous resta dans un premier temps à penser à une conception d'implémentation de la technologie Netchain étudiée au début du projet, par l'intermédiaire du P4. Dans un second temps, il nous fallait aussi penser à une nouvelle architecture, combinant DistBlockNet et Netchain, de sorte à implémenter ce dernier avec la notion de flow table dans la technologie Disblocknet. Au lieu de faire la reconfiguration par un seul contrôleur comme le fait SDN, on opte pour un contrôle décentralisé (par plusieurs contrôleurs et nœuds).

A la mi-janvier, synonyme de nouveau semestre, on a repris le travail sur le projet. On a commencé par créer l'architecture sur laquelle allait se baser nos tests futures, dont le premier test était d'implémenter une commutation au niveau 2 sur plusieurs switches et de réussir à établir une communication entre 2 hôtes distants en passant par ce réseau d'équipements.

Une fois cette étape réussie, il nous fallait passer à un routage au niveau 3, qui était plus délicat à implémenter cette fois-ci. Après concertation avec notre encadrant, elle nous a redirigé sur les forums github et autres qui nous ont beaucoup inspiré. Au final il y avait plusieurs facteurs à prendre en compte lors du routage d'un paquet.

Pour la prochaine étape, il nous fallait visualiser le contenu du trafic du réseau et voir les tables de routage et de commutation sur les switches, tâche assez ardue car les paramètres réseau telles que les adresses ip et les adresses mac étaient assez difficiles à déterminer en premier lieu, puis à les utiliser et les fixer dans un second temps.

En outre, après avoir fait une réunion avec notre encadrant, nous avons parlé de la partie data analytics de notre projet, et après avoir lu quelques documents de recherche, on s'est approprié la technique de monitoring utilisée dans ces papiers, puis on a réussi à la reproduire sur notre architecture, et ainsi voir finalement le contenu des paquets qui transitent dans le réseau et à imaginer des fonctions centrées sur des champs spécifiques de ces paquets.

Nous avons consacré les dernières semaines au test de notre architecture, à créer une base de données locale propre à chaque switch, contenant les champs de paquets qu'on a souhaité traiter, et au peaufinage du rapport.

Pour planifier les différentes étapes de notre projet, nous avons utilisé le diagramme de GANTT, la durée de notre projet est étalée entre octobre et mai comme le montre la figure ci-dessous :

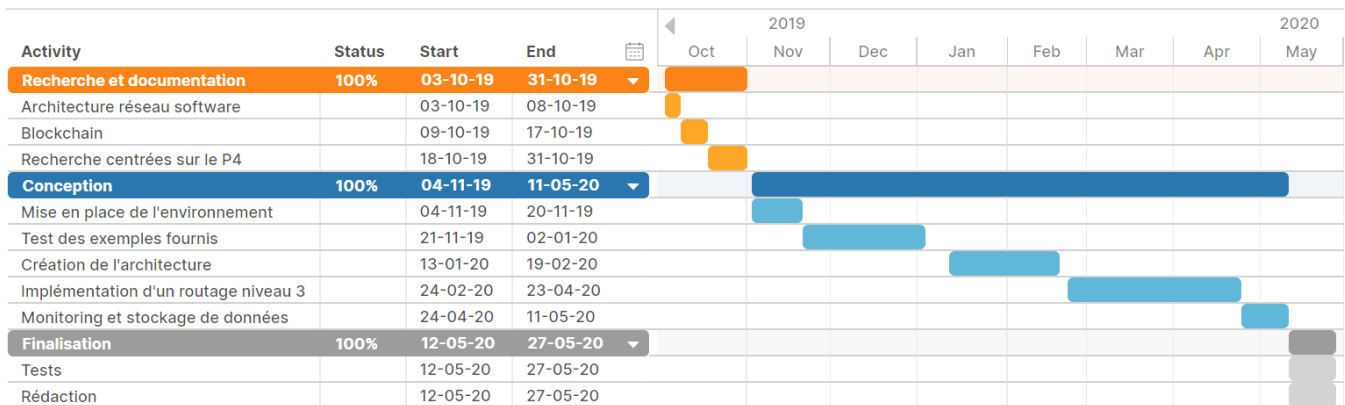


Figure 1 : Diagramme de GANTT

3 Bibliographie

- [1] P. Fernando et J. Wei, « Blockchain-Powered Software Defined Network-Enabled Networking Infrastructure for Cloud Management », 2019.
 - [2] G. Pujolle et O. Salvatori, *Les réseaux*, 9e édition édition 2018-2020. Eyrolles, 2018.
 - [3] Y. Qian *et al.*, « Towards decentralized IoT security enhancement: A blockchain approach », *Computers & Electrical Engineering*, vol. 72, p. 266-273, nov. 2018.
 - [4] <https://github.com/nsg-ethz/p4-learning/tree/master/slides> cours P4 ETH Zurich.
 - [5] S. Rathore, B. Wook Kwon, et J. H. Park, « BlockSecIoTNet: Blockchain-based decentralized security architecture for IoT network », *Journal of Network and Computer Applications*, vol. 143, p. 167-177, oct. 2019.
 - [6] P. K. Sharma, S. Singh, Y.-S. Jeong, et J. H. Park, « DistBlockNet: A Distributed Blockchains-Based Secure SDN Architecture for IoT Networks », *IEEE Communications Magazine*, vol. 55, n° 9, p. 78-85, sept. 2017.
- Article de revue scientifique et spécialisée, publiée par IEEE. Cet article expose une technologie qui combine les avantages des deux technologies, SDN et blockchains, pour former une nouvelle architecture distribuée et sécurisée. Les auteurs sont des chercheurs et docteurs, comptant plusieurs publications et référencés plusieurs fois sur Google scholar par exemple. Cette source est l'une des bases les plus solides sur laquelle on initiera notre projet.
- [7] K. g. Srinivasa, G. M. Siddesh, et H. Srinidhi, *Network data analytics : a hands-on approach for application development*. Springer, 2018.
 - [8] J. William, *Blockchain : the simple guide everything you need to know*. [Createspace], 2016.
 - [9] X. Xu Informaticien, I. Weber, et M. Staples, *Architecture for blockchain applications*. Springer, 2019.
 - [10] A. Yazdinejad, R. M. Parizi, A. Dehghantanha, et K.-K. R. Choo, « P4-to-blockchain: A secure blockchain-enabled packet parser for software defined networking », *Computers & Security*, vol. 88, p. 101629, janv. 2020.

C'est un article issu du journal *Computers & Security*, très bonne source spécialisée sur les nouveautés IT et Cybersécurité. Il a été écrit par quatre enseignants chercheurs exerçant dans différentes universités américaines et canadiennes. Il y est proposé une nouvelle architecture d'analyseur de paquets basé sur blockchains et implémenté sur SDN, et surtout programmé en P4, langage qu'on compte utiliser pour réaliser notre simulation.

[11] Zijun Hang, Mei Wen, Yang Shi, et Chunyuan Zhang, « Programming Protocol-Independent Packet Processors High-Level Programming (P4HLP): Towards Unified High-Level Programming for a Commodity Programmable Switch », *Electronics*, n° 9, p. 958, 2019.

[12] « NetChain: Scale-Free Sub-RTT Coordination (Extended Version) ». [En ligne]. Disponible sur: <https://arxiv.org/abs/1802.08236>. [Consulté le: 15-nov-2019].

Il s'agit d'un working paper, provenant de la plateforme arXiv, où sont diffusés des résultats de recherche, qui sont souvent plus détaillés qu'un article paru dans une revue scientifique. Ce document présente une nouvelle approche au sein du réseau en utilisant des switchs programmables à la place des habituels serveurs. Les auteurs sont tous des enseignants chercheurs spécialisés dans le domaine de l'informatique, pratiquant dans d'imminentes universités des Etats Unis et d'Italie. Ils ont à leur actif bon nombre de publications sur les technologies des réseaux. De ce fait, cette source se voit très pertinente et sera utile pour la réalisation de notre projet.

[13] « Secure Software-Defined Networking Based on Blockchain ». [En ligne]. Disponible sur: <https://arxiv.org/abs/1906.04342>. [Consulté le: 15-nov-2019].

[14] A. Khandelwal, R. Agarwal, et I. Stoica, « Confluo: Distributed Monitoring and Diagnosis Stack for High-speed Networks », p. 15.

4 Analyse

4.1 Objectif

Le but de notre projet est d'améliorer la récupération des données du trafic réseau, en optant pour une architecture SDN-Blockchain, déployée par l'intermédiaire de switchs.

Grâce à un pareil dispositif, on aura à tout moment T une vue globale sur le réseau, revoyant toutes les actions passées enregistrées dans un historique, et même planifier l'avenir et voir les conséquences d'une nouvelle instruction avant même de l'appliquer.

4.2 Spécifications des besoins fonctionnels

Les spécifications fonctionnelles ont pour objectif de décrire précisément l'ensemble des fonctions d'un logiciel ou d'une application, et de fixer ainsi le périmètre fonctionnel du projet.

Pour mener à bien donc notre projet, il nous a fallu une machine virtuelle contenant un

système d'exploitation Ubuntu, contenant les outils nécessaires suivants :

- **P4** : c'est un langage de haut niveau pour la programmation de protocoles indépendants de traitement de paquets, il travaille en coordination avec les protocoles de contrôle SDN semblables à Openflow. Grâce à P4 on va pouvoir contrôler le comportement du protocole selon nos désirs.
- **Mininet** : c'est un logiciel qui va nous permettre de visualiser le réseau directement sur notre machine virtuelle. Grâce à Mininet on va avoir une vue d'ensemble sur la topologie réseau qu'on a créée et ainsi voir le comportement du protocole implémenté dans les nœuds du réseau.
- **JSON** : il va nous permettre de créer une topologie réseau avec des nœuds, des hôtes et des contrôleurs (nœuds switches), de créer des liens entre les équipements.
- **Python** : on fait appel à Python pour analyser le fichier de configuration, créer un réseau virtuel avec des hôtes et des switches P4 utilisant Mininet.
- **Wireshark** : utile pour analyser les données qui sont échangées entre les nœuds.

4.3 Démonstration

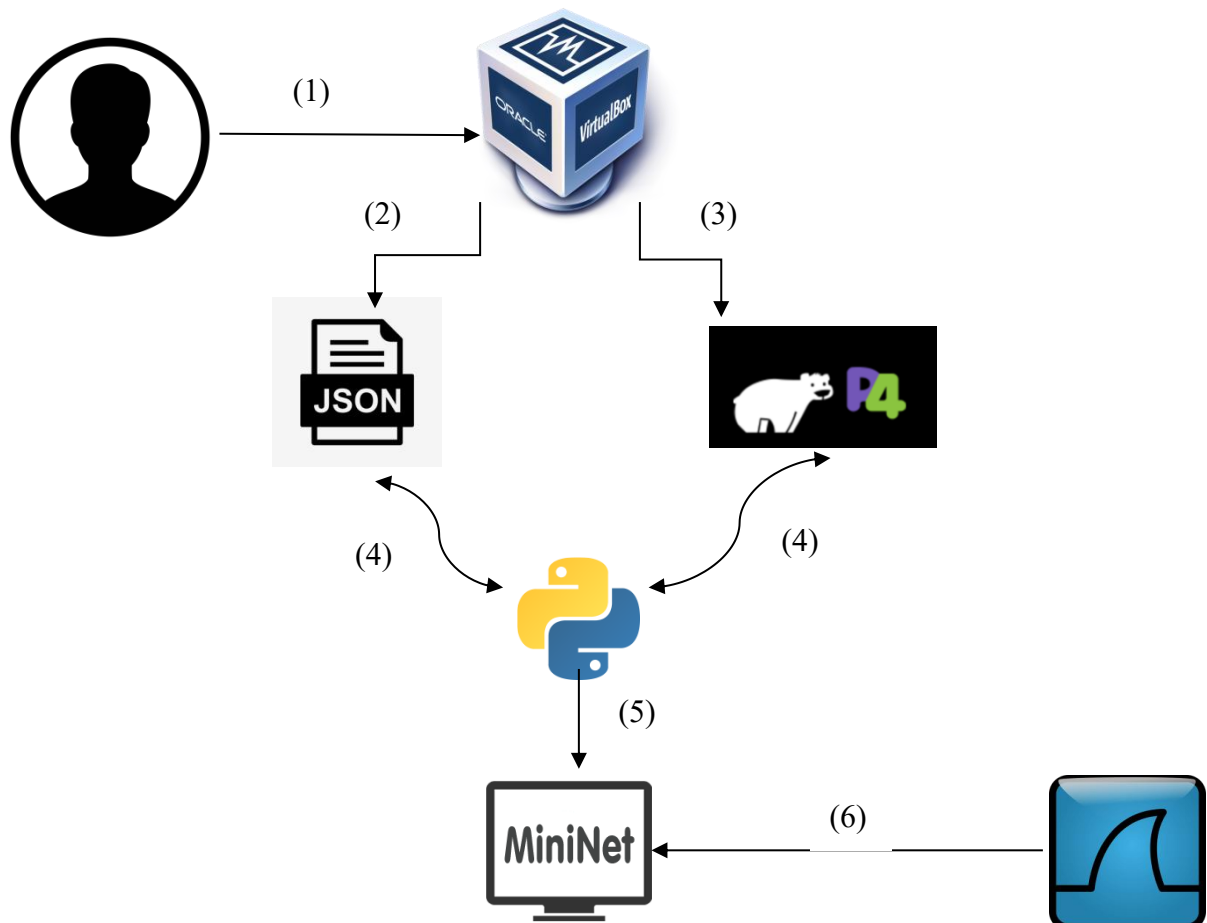


Figure 2 : Démonstration des applications utilisées

- (1) L'utilisateur se connecte sur la machine virtuelle qui tourne sous Ubuntu 16.4
- (2) Description topologique du réseau souhaité dans un fichier JSON
- (3) Description du protocole réseau à implémenter dans les nœuds du réseau (contrôleurs et switches)
- (4) Python lance le script contenant la configuration du comportement du réseau
- (5) Le résultat combinant la configuration de la topologie et le comportement du protocole programmé est affiché sur Mininet
- (6) Wireshark sert à faire la collecte des données sur l'une des interfaces des nœuds du réseau

4.4 Contraintes

La programmation de protocole dit indépendant en langage P4-16 est à ce jour pas très connue, il n'y a pas beaucoup de ressources sur lesquelles on peut se baser hormis le cours de l'université d'ETH Zürich « Advanced Topics in Communication Networks ».

De plus on se doit de programmer chaque protocole vu dans la partie 6.1 en langage P4 et essayer de les combiner au sein d'un même switch contrôleur puis de partager cela sur plusieurs switches pour obtenir une architecture de Blockchain.

Au début, pour réussir une commutation de paquet, nous avons assez d'exemples pour aboutir à ce qu'on voulait faire, mais par la suite, en voulant combiner plusieurs spécificités réseau, réussir à combiner un routage et une commutation tout en respectant l'intégrité des paquets entrants et sortants des switches, comme le calcul du checksum par exemple, a rendu la tâche rude. On s'est heurté donc à plusieurs difficultés, et un manque d'exemples pratiques s'est fait ressentir.

En outre, à chaque fois que l'on souhaite générer une topologie, les adresses ip se génèrent à partir des adresses mac d'une manière aléatoire à cause du fonctionnement fondamental du mininet.

Aussi, on a essayé d'implémenter un serveur web apache. Après avoir réussi un monitoring sur le trafic réseau avec des tests de ping, nous avons ensuite souhaité visualiser les champs numéros de séquence et acquittements au niveau de la couche transport, on a rencontré un problème de connexion. En effet, on s'est heurté à un problème entre notre topologie dans mininet et le branchement à internet, on a pas réussi à installer le serveur.

5 Conception

5.1 Développement

À l'entrée d'un paquet dans un nœud du réseau, on commence par l'analyser et séparer ses champs d'entêtes des différentes couches réseau (Ethernet, IP et TCP/UDP) pour un traitement futur [4].

Par la suite et selon ce qu'on désire avoir comme comportement du protocole, on va traiter les données d'en-têtes et leurs faire des modifications selon une table d'action spécifique au protocole indépendant qui sera créé [4].

Enfin, on réassemble les en-têtes modifiés grâce au deparser, le paquet en ressort modifié en direction du prochain nœud où il sera traité de la même manière [4].

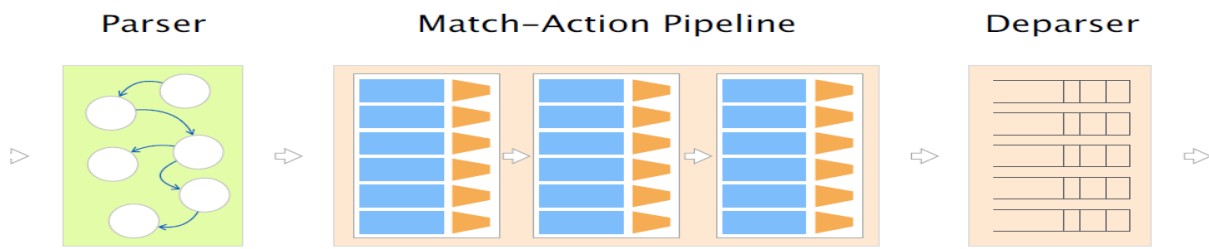


Figure 3 : Méthode de programmation d'un protocole indépendant pour le traitement de paquets [4]

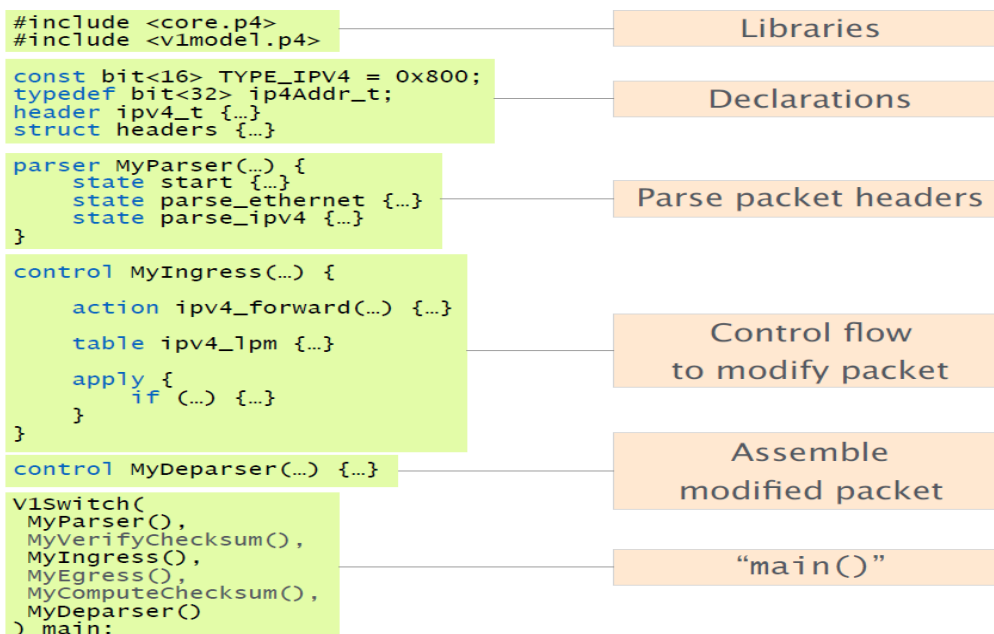


Figure 4 : Structure générale de code P4 [4]

Coté programmation P4, on a une vue de bloc semblable à la figure (4), on fait d'abord appel à des bibliothèques, puis on déclare le type de nos variables locales et globales, enfin on ajoute des fonctions qui vont séparer et analyser les paquets à l'entrée d'un nœud et les réassemble modifiés en sortie. La fonction de contrôle de flux va faire des modifications sur les en-têtes selon le comportement du protocole qu'on désire avoir [4].

5.2 Implémentation d'un routage niveau 3 et d'un monitoring sur le réseau

5.2.1 Création de la topologie

On a créé au début un fichier .JSON pour définir notre architecture réseau sur Mininet, elle est composée de deux hôtes et six switches.

```
26 "topology": {
27   "assignment_strategy": "l3",
28   "links": [[["h1", "s1"], ["h3", "s1"], ["h2", "s6"], ["s1", "s2"], ["s1", "s3"], ["s1", "s4"], ["s1", "s5"], ["s2", "s6"],
29     ["s3", "s6"], ["s4", "s6"], ["s5", "s6"]],
30   "hosts": {
31     "h1": {
32       },
33     "h2": {
34       },
35     "h3": {
36       }
37   },
38   "switches": {
39     "s1": {
40       },
41     "s2": {
42       },
43     "s3": {
44       },
45     "s4": {
46       },
47     "s5": {
48       },
49     "s6": {
```

Figure 5 : Description de la topologie en .json

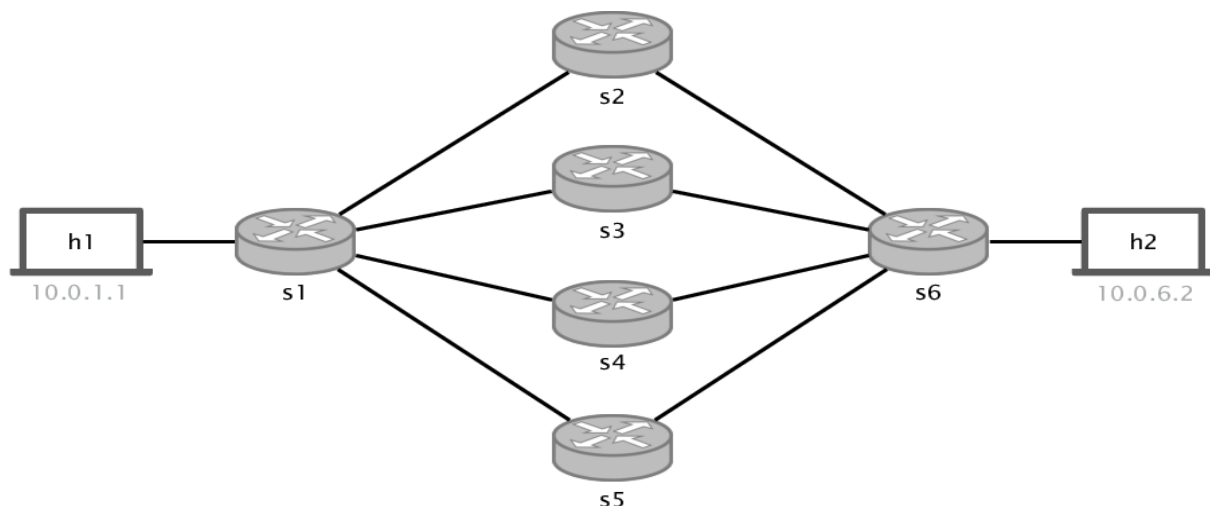


Figure 6 : Vue globale de la topologie

5.2.2 Création de fichier P4

On définit les en-têtes de notre trame, c'est-à-dire l'en-tête ethernet, ip et tcp avec les tailles standards de chaque en-tête. Comme le langage P4 permet de créer nos propres en-têtes personnalisés, utilisables dans des fonctions ultérieures pour le monitoring, on choisit de créer une structure comprenant plusieurs paramètres, tels que les adresses ip source et destination, TTL, numéros de port et numéros de séquence etc...

```
18 ▾ header ethernet_t {  
19     macAddr_t dstAddr;  
20     macAddr_t srcAddr;  
21     bit<16> etherType;  
22 }  
23 // les en-têtes ethernet et ip avec les standards  
24 ▾ header ipv4_t {  
25     bit<4> version;  
26     bit<4> ihl;  
27     bit<6> dscp;  
28     bit<2> ecn;  
29     bit<16> totalLen;  
30     bit<16> identification;  
31     bit<3> flags;  
32     bit<13> fragOffset;  
33     bit<8> ttl;  
34     bit<8> protocol;  
35     bit<16> hdrChecksum;  
36     ip4Addr_t srcAddr;  
37     ip4Addr_t dstAddr;  
38 }
```

Figure 7 : Structure des en-têtes standards

```
59 //un en-tête personnalisé avec les champs qu'on souhaite monitorer  
60 ▾ struct learn_t {  
61     bit<8> digest;  
62     bit<16> etherType_moni;  
63     bit<8> ttl_moni;  
64     bit<8> protocol_moni;  
65     bit<32> srcIP;  
66     bit<32> dstIP;  
67     bit<48> macSRC_moni;  
68     bit<48> macDST_moni;  
69 }
```

Figure 8 : Structure de l'en-tête personnalisé

Par la suite, on analyse les paquets entrants, tout en extrayant les en-têtes des différentes couches :

```
100
101 }
102 // on commence par analyser le paquet entrant dans un switch
103 state parse_ethernet {
104 // on extrait l'en-tête ethernet et on vérifie qu'on a bien de l'ipv4
105     packet.extract(hdr.ethernet);
106     transition select(hdr.ethernet.etherType){
107         TYPE_IPV4: parse_ipv4;
108         default: accept;
109     }
110 }
111 // on extrait l'en-tête ip et on vérifie qu'on a du tcp sur la couche supérieure
112 state parse_ipv4 {
113     packet.extract(hdr.ipv4);
114     transition select(hdr.ipv4.protocol){
115         6 : parse_tcp;
116         default: accept;
117     }
118 }
119 // et enfin on extrait l'en-tête tcp
120 state parse_tcp {
121     packet.extract(hdr.tcp);
122     transition accept;
123 }
```

Figure 9 : Analyse des en-têtes en entrée du switch

Dans la partie traitement des en-têtes, on implémente un routage de niveau 2 et 3, où les champs mac et ip vont prendre les adresses fournies par le contrôleur. On implémente aussi une fonction de hachage ECMP afin d'éviter les boucles de routage sur notre topologie.

```
145 //*****Partie pour le routage *****
146 //fonction de hashage pour ECMP
147 action ecmp_group(bit<14> ecmp_group_id, bit<16> num_nhops){
148     hash(meta.ecmp_hash,
149     HashAlgorithm.crc16,
150     (bit<1>)>0,
151     { hdr.ipv4.srcAddr,
152       hdr.ipv4.dstAddr,
153       hdr.tcp.srcPort,
154       hdr.tcp.dstPort,
155       hdr.ipv4.protocol},
156     num_nhops);
157
158     meta.ecmp_group_id = ecmp_group_id;
159 }
160 // fonction pour le routage L2/L3
161 action set_nhops(macAddr_t dstAddr, egressSpec_t port) {
162
163     //l'@-MAC source prends l'@ de la destination précédent
164     // ce n'est pas correcte mais ça permet de ne pas laisser le champ vide
165     hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
166
167
168     //l'@-MAC de destination prends l'adresse dans la table de correspondance fournit par le controleur
169     hdr.ethernet.dstAddr = dstAddr;
170
171     //Pareil pour le port de sortie
172     standard_metadata.egress_spec = port;
173
174     //on decremente le TTL par 1 lors d'un passage par un switch
175     hdr.ipv4.ttl = hdr.ipv4.ttl - 1;
176
177 }
```

Figure 10 : Routage Equal Cost Multi Path

À cela, on ajoute des valeurs à notre en-tête personnalisé créé précédemment, qui sera lu et traité par le contrôleur afin de générer une base de données locale à chaque switch contenant les en-têtes de paquets qui transitent dans le réseau.

```
177
178 //*****PARTIE MONITORING *****
179
180
181 // on génère un nombre aléatoire pour vérifier que le controleur arrive à lire les paquets via les sockets de connexion
182 random(meta.learn.digest,(bit<8> 0, (bit<8>) 255);
183
184
185 //On remplit les champs de notre en-tête personnalisé par une copie des champs correspondant
186 // des paquets en entré avant modification
187 meta.learn.etherType_moni=hdr.ethernet.etherType;
188 meta.learn.ttl_moni=hdr.ipv4.ttl;
189 meta.learn.protocol_moni=hdr.ipv4.protocol;
190 meta.learn.srcIP = hdr.ipv4.srcAddr;
191 meta.learn.dstIP = hdr.ipv4.dstAddr;
192 meta.learn.macSRC_moni=hdr.ethernet.srcAddr;
193 meta.learn.macDST_moni=hdr.ethernet.dstAddr;
194
```

Figure 11 : Remplir l'en-tête personnalisé avec les champs de monitoring

Une fois que le paquet est réassemblé et qu'on a vérifié son intégrité, il peut à présent transiter sans soucis. Coté contrôleur, en python cette fois, on fait en sorte de créer dynamiquement des tables de routage tout en intégrant la fonction `ecmp`, le routage de niveau 3 est donc effectué avec succès, et notre réseau converge.

Pour la partie de monitoring, le contrôleur va créer des sockets de connexion, et ainsi pouvoir lire les paquets qui passent par les switchs, et extrait l'en-tête personnalisée émise précédemment pour faire du traitement de données, et écrire le contenu dans un fichier `.txt`, ce monitoring se fait sur un ou plusieurs switchs, selon la fonction qu'on souhaite réaliser.

```
154
155 def run_loop(self): #création de socket de connexion pour lire le contenu des paquets transitant dans le réseau
156
157     sub = npy.Socket(npy.AF_SP, npy.SUB)
158     notifications_socket = self.controller.client.bm_mgmt_get_info().notifications_socket
159     print "connecting to notification sub2 %s" % notifications_socket
160     sub.connect(notifications_socket)
161     sub.setsockopt(npy.SUB, npy.SUB_SUBSCRIBE, '')
162
163     while True:
164         msg = sub.recv()
165         self.recv_msg(msg)
166
167 def main():
168
169     MonitorController("s1").run_loop()
170
```

Figure 12.a : Création d'une socket de connexion

```
109 class MonitorController():
110     #Création du monitoring
111
112     def __init__(self, sw_name):
113
114         self.sw_name = sw_name
115         self.topo = Topology(db="topology.db") #le controleur va chercher les informations liée à la topologie du réseau
116         self.sw_name = sw_name #le nom des switches
117         self.thrift_port = self.topo.get_thrift_port(sw_name) #les ports de connexion
118         self.controller = SimpleSwitchAPI(self.thrift_port)
119
120     def recv_msg(self, msg):
121         f=open("storage.txt", "a") #ouvre un fichier texte comme base de données""
122
123         topic, device_id, ctx_id, list_id, buffer_id, num = struct.unpack("<iqiqi",
124                                     msg[:32])
125
126         #print num, len(msg)
127         offset = 13
128         msg = msg[32:]
129         for sub_message in range(num):
130
131             random_num, ethertype, ttl, protocol, src, dst = struct.unpack(("!BHHBII"), msg[0:offset]) #extraite les champs d'en-tête qu'on souhaite traiter
132             SRC_port=struct.unpack(("!H"), msg[13:15])
133             DST_port=struct.unpack(("!H"), msg[15:17])
134             MAC_src=struct.unpack(("!HI"), msg[17:23])
135             MAC_dst=struct.unpack(("!HI"), msg[23:29])
136
137             #print( type(MAC_src))
138
139             print "random number:", random_num, "ethertype", ethertype, "ttl", ttl, "protocole", protocol, "src ip:", str(ipaddress.IPv4Address(src)), "dst ip:",
140                 str(ipaddress.IPv4Address(dst))
141
142             y= str(datetime.datetime.now()), "random number:", random_num, "ethertype", hex(ethertype), "ttl", ttl, "protocole", protocol, "src ip:",
143                 str(ipaddress.IPv4Address(src)), "dst ip:", str(ipaddress.IPv4Address(dst)), "port source:", SRC_port, "port destination:", DST_port
144
145             msg = msg[offset:]
146             y=str(y) #transformer les données en text
147             f.write ( "*****Monitoring on sw1 *****" #écriture dans le fichier
148                 + "\n"+y+"\n")
```

Figure 12.b : Monitoring et stockage de données

5.2.3 Exécution de l'architecture

Après la création des fichiers JSON, P4 et python, les bouts se joignent et il nous reste alors que l'exécution. Cela se fait avec la commande « sudo p4run », cette commande va générer un script Python pour créer la topologie sur Mininet et envoyer le code P4 au switch, la figure suivante montre le résultat de cette commande :

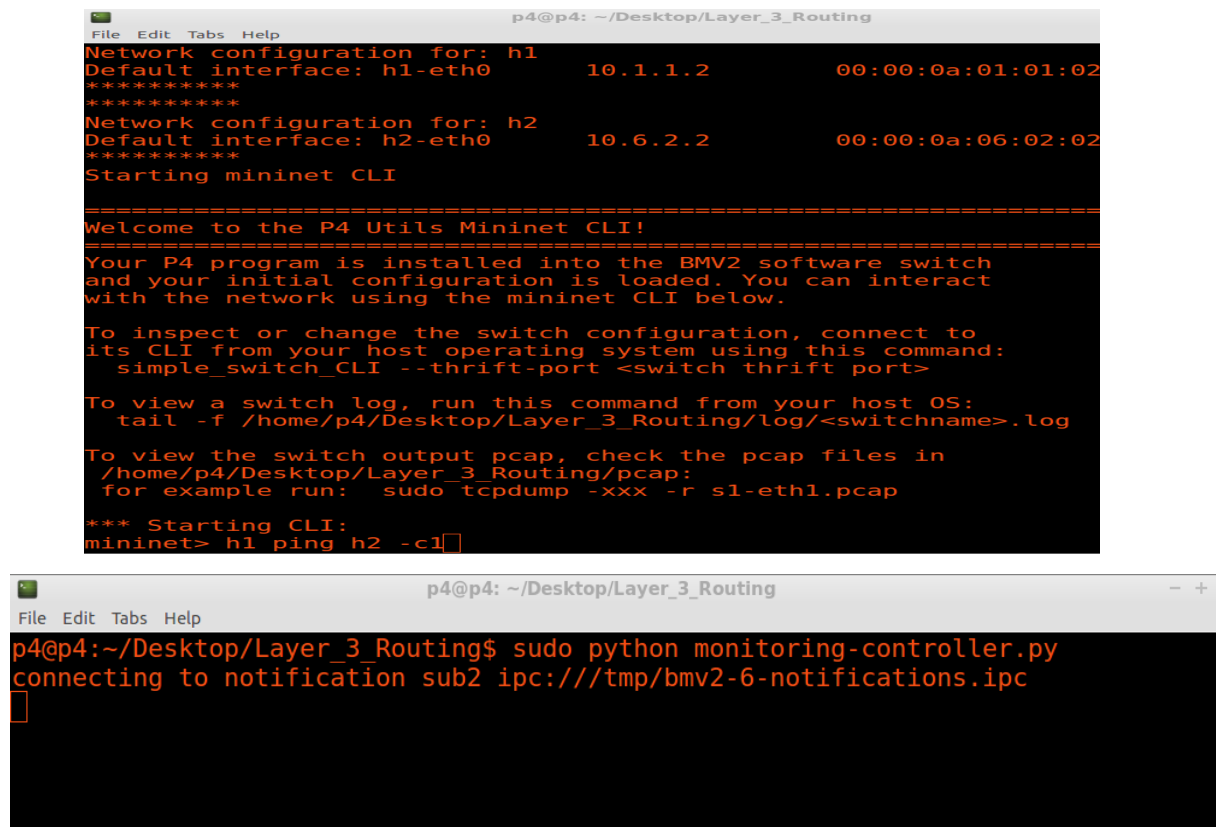

```
Saving mininet topology to database.
s1 -> Thrift port: 9090
s2 -> Thrift port: 9091
s3 -> Thrift port: 9092
s4 -> Thrift port: 9093
s5 -> Thrift port: 9094
s6 -> Thrift port: 9095
*****
Network configuration for: h1
Default interface: h1-eth0      10.1.1.2      00:00:0a:01:01:02
*****
*****
Network configuration for: h2
Default interface: h2-eth0      10.6.2.2      00:00:0a:06:02:02
*****
*****
Starting mininet CLI

=====
Welcome to the P4 Utils Mininet CLI!
=====
```

Figure 13 : Prompt mininet

5.2.4 Tests et résultats

Après convergence du réseau, on obtient une architecture qui marche et une base de données contenant les champs d'en-tête que l'on souhaite traiter



The image shows two screenshots of a terminal window. The top screenshot displays the mininet CLI prompt and network configuration for hosts h1 and h2. The bottom screenshot shows the user running a command to start the monitoring controller.

```
p4@p4: ~/Desktop/Layer_3_Routing
File Edit Tabs Help

Network configuration for: h1
Default interface: h1-eth0      10.1.1.2      00:00:0a:01:01:02
*****
*****
Network configuration for: h2
Default interface: h2-eth0      10.6.2.2      00:00:0a:06:02:02
*****
*****
Starting mininet CLI

=====
Welcome to the P4 Utils Mininet CLI!
=====

Your P4 program is installed into the BMV2 software switch
and your initial configuration is loaded. You can interact
with the network using the mininet CLI below.

To inspect or change the switch configuration, connect to
its CLI from your host operating system using this command:
  simple_switch_CLI --thrift-port <switch thrift port>

To view a switch log, run this command from your host OS:
  tail -f /home/p4/Desktop/Layer_3_Routing/log/<switchname>.log

To view the switch output pcap, check the pcap files in
/home/p4/Desktop/Layer_3_Routing/pcap:
for example run:  sudo tcpdump -xxx -r s1-eth1.pcap

*** Starting CLI:
mininet> h1 ping h2 -c1

p4@p4:~/Desktop/Layer_3_Routing$ sudo python monitoring-controller.py
connecting to notification sub2 ipc:///tmp/bmv2-6-notifications.ipc
```

```
p4@p4: ~/Desktop/Layer_3_Routing
File Edit Tabs Help

table_add at s5:
Adding entry to lpm match table ipv4_lpm
match key:      LPM-0a:06:02:02/24
action:         set_nhop
runtime data:   a6:bd:39:d5:de:39  00:02
Entry has been added with handle 1

table_add at s4:
Adding entry to lpm match table ipv4_lpm
match key:      LPM-0a:01:01:02/24
action:         set_nhop
runtime data:   ca:78:a0:43:76:88  00:01
Entry has been added with handle 0

table_add at s4:
Adding entry to lpm match table ipv4_lpm
match key:      LPM-0a:06:02:02/24
action:         set_nhop
runtime data:   c2:d0:9e:12:81:ed  00:02
Entry has been added with handle 1

connecting to notification sub2 ipc:///tmp/bmv2-1-notifications.ipc
```

Figure(s) 14 : Phase de test

```
*storage.txt
File Edit Search Options Help

*****monitoring on SW6 *****
('2020-06-02 15:48:31.517985', 'random number:', 207, 'ethertype', '0x800', 'ttl', 61, 'protocole', 1, 'src ip:', '10.1.1.2', 'dst ip:', '10.6.2.2')
*****monitoring on SW1 *****
('2020-06-02 15:48:31.518513', 'random number:', 194, 'ethertype', '0x800', 'ttl', 63, 'protocole', 1, 'src ip:', '10.1.1.2', 'dst ip:', '10.6.2.2')
*****monitoring on SW6 *****
('2020-06-02 15:48:31.518801', 'random number:', 140, 'ethertype', '0x800', 'ttl', 63, 'protocole', 1, 'src ip:', '10.6.2.2', 'dst ip:', '10.1.1.2')
*****monitoring on SW1 *****
('2020-06-02 15:48:31.521090', 'random number:', 181, 'ethertype', '0x800', 'ttl', 61, 'protocole', 1, 'src ip:', '10.6.2.2', 'dst ip:', '10.1.1.2')

p4@p4: ~/Desktop/Layer_3_Routing
File Edit Tabs Help

table_add at s4:
Adding entry to lpm match table ipv4_lpm
match key:      LPM-0a:01:01:02/24
action:         set_nhop
runtime data:   ca:78:a0:43:76:88  00:01
Entry has been added with handle 0

table_add at s4:
Adding entry to lpm match table ipv4_lpm
match key:      LPM-0a:06:02:02/24
action:         set_nhop
runtime data:   c2:d0:9e:12:81:ed  00:02
Entry has been added with handle 1

connecting to notification sub2 ipc:///tmp/bmv2-1-notifications
random number: 194 ethertype 2048 ttl 63 protocole 1 src ip: 10
.6.2.2
random number: 181 ethertype 2048 ttl 61 protocole 1 src ip: 10
.1.1.2
```

Figure 15 : Résultat du monitoring

5.3 Les protocoles indépendants à implémenter

5.3.1 Stockage des Clés-Valeurs :

Comme les switches programmables ont des registres pour stocker des données utilisateurs, le protocole sépare les clés et les valeurs, puis les stocke dans un tableau, chaque hôte a un agent propre au protocole qui va faire des requêtes au nœud contrôleur pour obtenir une règle de transmission du flux, il cache la clé-valeur donc par rapport à l'application. De là on peut réutiliser ces données pour créer des fonctions qui vont par exemple lever une alarme lorsqu'un paquet est transmis avec un champ d'en-tête spécifique comme une certaine valeur de TTL, des numéros de séquence TCP qui se répètent etc...

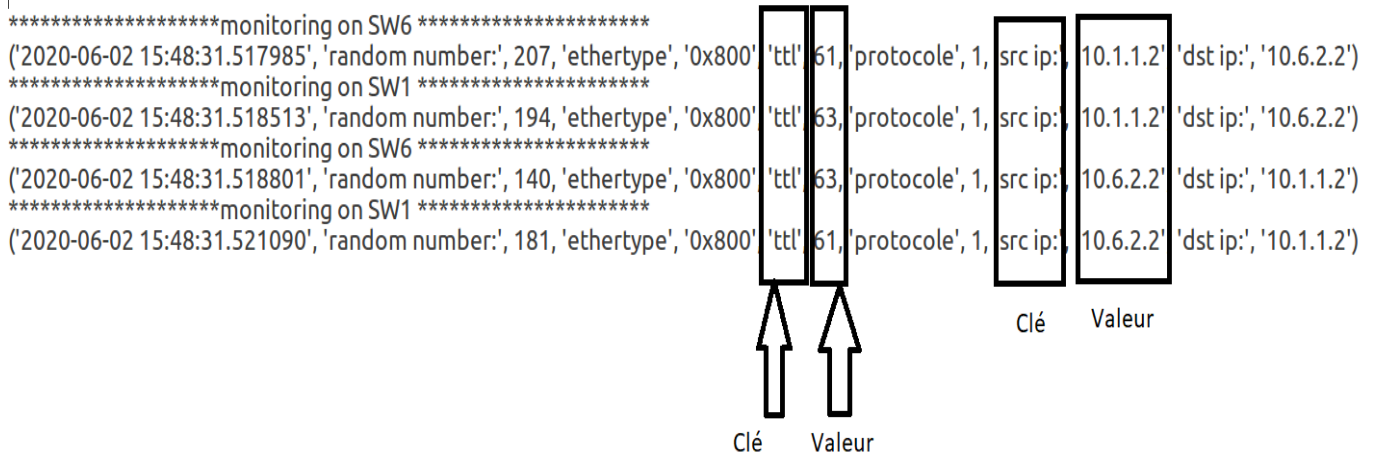


Figure 16 : Stockage des clés-valeurs

5.3.2 En-tête modifié :

En langage p4 nous pouvons faire de la description protocolaire très détaillée, on peut imposer le format de nos paquets et de nos trames et même ajouter des champs spécifiques qui vont servir pour le traitement de données, ces champs-là ne sont pas amenés à être modifiés lors d'un passage par un switch, dès lors on peut faire de la collecte de données à tout moment dans le réseau.

Dans notre cas et pour faciliter l'implémentation, on choisit d'ajouter un en-tête Ethernet pour le routage niveau 2 (commutation), l'en-tête IP pour le routage niveau 3 , l'en-tête TCP pour simuler un serveur web (des requêtes http) et enfin un en-tête personnalisé contenant différents champs pour le monitoring et la collecte de données. Cet en-tête sert à implémenter plusieurs principes, routages de segments ordonnés et à définir des opérations selon les requêtes clés-valeurs [12].

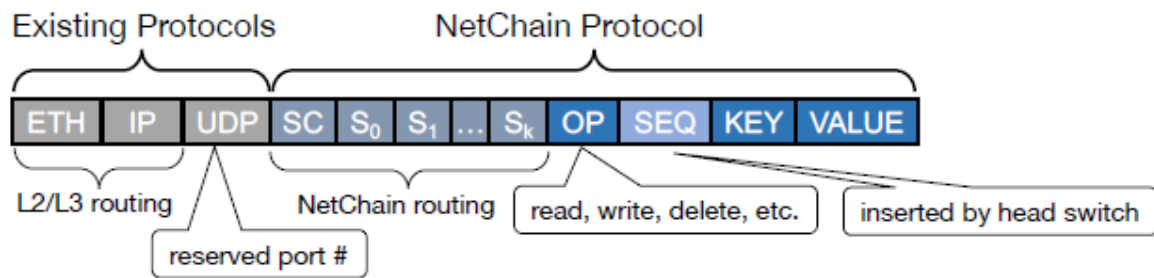


Figure 17 : En-tête du segment modifié

5.3.3 Routage selon une structure en chaîne

5.3.3.1 Chain replication

Dans une architecture basique où on a des nœuds présents que pour la redondance (primary backup), l'implémentation était trop lourde sachant la capacité des switches asics (Cisco 2960 par exemple). En optant pour la variante utilisée par Netchain, nous avons alors créé une topologie à six switches et deux hôtes. Seul soucis, lors du routage, les switches ne savent pas router les paquets sur deux chemins avec des coûts égaux, du moins il faut programmer ce comportement au sein des switches. Dès lors, nous avons opté pour l'ECMP (Equal Cost Multi Path), lorsque des paquets arrivent au sein d'un switch, on prend les en-têtes et on en déduit une valeur de Hash, cette valeur de hash va correspondre avec la table fournie par le contrôleur, ainsi on obtient un tuple hash-port de sortie. On arrive donc à commander le switch l'action qu'il doit faire lorsqu'il a des coûts de liens égaux vers une destination.

5.3.3.2 Routage implémenté

Le principe de routage des requêtes dans une structure de réplification de chaîne est simple et comme suit [12] :

- Chaque hôte se voit attribué une adresse ip unique dans le réseau à partir de son adresse mac
- Un routage au niveau 2 (Commutation) peut se faire de manière statique où nous entrons en ligne de commande de L'API du switch les adresses mac destinations et leurs ports de sortie correspondants, ou bien dans un fichier texte ajouté au .JSON.
- Un routage au niveau 3 permet un hôte de joindre une adresse distante dans le réseau, comme notre architecture est basée sur SDN où le plan de données et de contrôle sont séparés, le contrôleur va fournir une table de routage au switch lorsque ce dernier souhaite transmettre un paquet, qui sera encapsulé par les champs Ethernet destination et port de sortie sans oublier de décrémenter le TTL comme dans la figure 11 vu plus haut.

Grâce à un routage pareil et des numéros de séquence attribués aux segments, on peut éviter de se heurter au « Best-effort » d'UDP et ainsi ne pas avoir de déséquencement.

6 Compte rendu

Notre objectif consiste à faire de la récupération de données de trafic réseau à tout instant et ceci avec une granularité suffisante en optant pour une architecture SDN dans un réseau de blockchain déployé grâce à des switches programmables.

Pour réussir à implémenter une technologie dans des switches, nous devons d'abord penser à programmer différents protocoles réseaux qui vont nous garantir [12] :

- Un stockage d'objets au sein des switches (clés-valeurs)
- Un routage des paquets selon une structure en chaîne

De plus, on se doit de trouver un moyen de généraliser ces protocoles sur quelques nœuds contrôleurs dans le réseau, donc décentraliser le rôle du contrôleur et le partager en plusieurs switches [6]. Contrairement à la technique Blockchain où chaque switch se devait de garder une base de données locale puis de la vérifier avec ses voisins proches, on s'est plutôt penché sur une base de données commune, sur laquelle des contrôleurs décentralisés contribuent à remplir cette base.

6.1 Approche prise

À partir d'une architecture similaire à la nôtre, on peut ainsi récupérer les données du trafic du réseau à tout moment

6.1.1 Pourquoi des switches programmables ?

Dans un réseau distribué tel que SDN où le plan de données et le plan de contrôle sont séparés, le rôle du contrôleur est généralement pris par un serveur (donc un software).

Dans notre cas, on opte pour une approche au sein du réseau pour avoir une coordination rapide entre les nœuds, et ceci grâce à des switches programmables. Ces switches programmables (Barefoot tofino) vont nous donner accès à un débit en sortie très élevé, comme la coordination se fait dans le réseau où tous les switches font tourner un protocole de consensus, on obtiendra un temps de traitement de requêtes avoisinant le demi RTT [12].

Le stockage des tables de flux semblables à celles de SDN seront cette fois-ci stockées dans les switches programmables, qui ont une capacité de stockage plus élevée que les switches ordinaires. Ainsi les règles de commutations seront vite transmises entre les nœuds comme la coordination se fait au sein du réseau [12].

De plus, on peut assurer une forte consistance et une tolérance aux fautes en modifiant la

structure des nœuds du réseau en une structure en chaîne qui va permettre de traiter les paquets d'une façon différente, les requêtes d'écriture vont passer de nœud en nœud jusqu'au bout de la chaîne et les requêtes de lecture vont passer quant à elle directement au dernier nœud [12].

6.1.2 Pourquoi une architecture Blockchain ?

Blockchain est une technologie qui fonctionne sans organe central et qui constitue une base de données des échanges du trafic réseau, on opte pour une telle architecture pour pouvoir partager cette fois les tables de flux qui étaient propres à un seul contrôleur (dans SDN) entre quelques nœuds du réseau (des switchs programmables) ce qui va permettre de décentraliser le contrôleur [6].

Au lieu d'avoir une seule entité qui gère la commutation dans le réseau, on répartie cette tâche sur quelques nœuds qui vont s'entraider à gérer le flux du réseau et ainsi augmenter ses performances de qualité de services [6].

Un débit élevé et un temps de réponse écourté sont garantis par les performants switchs programmables et une architecture décentralisée qui permet de partager les l'informations entre les nœuds du réseau, vont nous permettre de récupérer les données du trafic réseau à une granularité suffisante.

6.2 Monitoring et stockage de données

Le contrôleur étant séparé du plan de données nous permet d'avoir une vue externe au switch. Lors d'un traitement de paquet par un switch, le contrôleur va créer une socket de connexion avec le switch qui va lui permettre de lire le contenu des en-têtes du paquet [figure 12.a], d'en extraire les champs souhaités et de les stocker dans une base de données avec une entrée et une valeur (Key-Value). [Figure 16]

6.2.1 Fonctions basées sur les données du monitoring

Une fois une base de données contenant des informations utiles sur le trafic obtenue, nous pouvons à présent créer des fonctions intéressantes et dynamiques liées au comportement et aux performances de notre réseau.

Une première fonction serait par exemple, lorsqu'un paquet est rerouté à cause d'une mise en échec ou lors d'un bug, son parcours dans le réseau (source et destination) contenu dans les en-têtes permet de lever une alerte [14] et ainsi d'indiquer le point qui pose problème au sein du réseau.

Une seconde fonction permettrait de connaître le nombre de saut entre une source et une destination en analysant le champ TTL, et si ce dernier prend une valeur incohérente par

rapport au nombre de switch dans notre réseau, on en déduit qu'il y'a une erreur de routage comme une boucle ou un trou noir et ainsi pouvoir analyser et remédier à cette erreur.

Une dernière fonction serait liée au flux TCP, le contrôleur filtre les paquets contenant un numéro de séquence TCP plus petit que le champ SEQ et un timestamp plus large que TS, probablement causé par des retransmissions [14] et ainsi peut rééquilibrer la valeur de la fenêtre de congestion et le seuil SStresh.