

---

**POLYTECHNIQUE  
MONTREAL**

UNIVERSITÉ  
D'INGÉNIERIE



---

# Revue de littérature

2023-2024

École Polytechnique de Montréal

Dernière mise à jour: 16 février 2023

---

Yann **Roberge**

1802531

---

## 1 Sujets par mots-clefs

(Remplir plus tard quand il y aura 75 références)

## 2 Résumés de lecture

### 2.1 P4 : programming protocol-independent packet processors, [1], Bosshart et al., 2014

**Pertinence** : 3 (article fondateur) **Sujets** : langage P4, SDN

**Problématique** : Comment étendre les fonctionnalités des SDN sans rajouter une complexité hors de contrôle à Openflow ?

**Prérequis** : MPLS : [2], [3] ; PortLand : [?], [?] ; VLANs, dot1Q : [4], [5] ; champ EtherType : [6]. Les trois sont utilisés à titre d'exemple. Openflow.

**Réponse** : Arrêter d'ajouter toujours plus d'entêtes possibles à la spécification d'Openflow. Implémenter à la place un 'Openflow 2.0' qui soit indépendant du protocole, permette la reconfiguration du plan de données, et soit indépendant de la cible.

**Résumé** : Les auteurs proposent le langage P4 comme Openflow 2.0. Le langage spécifié doit trouver un équilibre entre l'exhaustivité de son support de protocoles et sa simplicité. Les développeurs P4 devraient pouvoir développer des modèles indépendant de la cible de la même manière que les développeurs C peuvent développer des programmes sans se soucier du jeu d'instructions abstrait par le compilateur.

#### 2 - Modèle abstrait

Les programmes suivront un modèle standard de commutateur formé d'une chaîne amorcée par un parseur (FSM), suivi d'une série de table *match-action* et un désérialiseur (Le modèle qui allait devenir PISA.). Les auteurs font la distinction entre les opérations de *configuration* et *peuplement* (*populating*, des tables), depuis renommées plan de données et plan de contrôle. Ils définissent les métadonnées comme information passée d'une table à une autre - port d'entrée, *timestamp*, etc. L'AQM fonctionne comme sous Openflow : des tables d'actions distribuent les paquets entre les ports de sortie ; ils prévoient que l'on puisse ajouter d'autres fonctions de contrôle de congestion au langage.

#### 3 - A programming language

Le P4 doit pouvoir définir le format de chaque entête. Des alternatives au P4 seraient Click, et Openflow 1.0 lui-même. Le premier décrit le traitement de paquets avec une syntaxe C++ et vise l'implémentation de commutateurs logiciels. Il est difficile à synthétiser en matériel parce qu'il n'exprime pas le traitement sous forme de pipeline de flux de contrôle. Openflow 1.0 ne permet pas de définir librement des entêtes.

#### 4 - P4 by example

Ils utilisent l'exemple des étiquettes MPLS pour définir les structures et mot-clefs de base qui

sont toujours dans le  $P4_{16}$  aujourd'hui :

**Headers** champs d'entêtes, définition de leurs valeurs admissibles

**Parsers** Machine d'identification d'entêtes (et de vérification)

**Tables** Traitements à appliquer en fonction des entêtes & actions

**Controls** Blocs de contrôle de flux, menant d'une table à une autre.

Les tables interagissent avec le plan de contrôle SDN. Elles sont synthétisées sur les ressources de la cible : RAM pour les correspondances exactes (*exact-match*), TCAM pour les correspondances ternaires (et pour les LPM je suppose ?). La compilation est 'facile' pour des cibles logicielles, moins pour les ASICs. Le compilateur *back-end* traduit la représentation parsée vers la cible matérielle.

**Avis :** Le papier pose les bases du langage P4 et de la métaarchitecture PISA tels qu'on les connaît aujourd'hui. À la base, l'objectif du langage était simplement de dépasser le manque de flexibilité intrinsèque des SDN; alors qu'aujourd'hui on met plus en avant les applications supplémentaires (INT, in-network computing, etc.) possibles sur les commutateurs P4. La syntaxe a beaucoup changé depuis 2014, surtout la FSM du parseur et les tables; mais toutes les idées de base se retrouvent déjà dans ce papier. La possibilité de synthèse sur FPGA n'est pas même évoquée dans ce papier. Il faut croire que les choses ont changées depuis puisque je me souviens qu'à la keynote P4 de Mai 2022 M. McKeown affirmait que les commutateurs sont aujourd'hui plus rapides sur matériel reconfigurable que sur ASIC, parce que l'implémentation FPGA n'est pas obligée de s'encombrer de tout un tas de fonctionnalités niches dont la plupart ne seront jamais utilisées.

## 2.2 Oracle VirtualBox User Manual, [7], Oracle, 2004-2023

**Pertinence** : 3 ; tous les environnements P4 et réseau en général passent par des VMs, donc autant savoir ce qu'on fait. **Sujets** : environnements de développements

**Problématique** : Comprendre les réglages et les configurations réseaux possibles avec les VMs, dans le but d'arriver à monter facilement de nouveaux environnements.

**Prérequis** : Aucun

**Réponse** : Arrêter d'ajouter toujours plus d'entêtes possibles à la spécification d'Openflow. Implémenter à la place un 'Openflow 2.0' qui soit indépendant du protocole, permette la reconfiguration du plan de données, et soit indépendant de la cible.

**Résumé** :

- Section 4 : Configuring VMs. Ils passent à travers tous les réglages possibles du matériel. En particulier les réglages possibles des interfaces débogage sérielles. La plupart sont difficiles à utiliser parce que VBox n'a pas accès aux fichiers `/dev/ttyS*` de l'hôte.
- Section 7 : Virtual networking. Ils passent à travers tous les réglages des cartes réseau virtuelles : NAT, bridge, host NAT, internal network, d'autres accessibles hors du GUI.

On peut contrôler VBox entièrement par CLI avec VBoxManage. Certaines fonctionnalités avancées ne sont accessibles que par ligne de commande, pas par le GUI.

Troubleshooting : s'ajouter soi-même au groupe UNIX `vboxusers`.

**Avis** : En règle générale la configuration de VM se comprend très bien par analogie avec le montage de PC physique (insérer des cartes dans la carte mère), et les configurations réseau se comprennent comme des montages réseau avec des switches et des routeurs physiques.

## 2.3 P4 Network Programming Language—what is it all about ?, [8], Pawel, 2020

**Pertinence** : 3 (article d'introduction au P4) **Sujets** : langage P4

**Problématique** : Quels nouvelles fonctionnalités réseau le P4 débloquent-il ? Quel est le potentiel d'usages du langage ?

**Prérequis** : Aucun

**Réponse** : Décharger des couches de la pile réseau normalement gérées en CPU vers le NIC ou les commutateurs/routeurs : couche IP, TCP, NAT, pare-feu, etc. Simuler des designs de commutateurs ; implémentation de nouveaux commutateurs logiciels. L'usage le plus prometteur est la possibilité de collecte de données de télémétrie directement sur les équipements réseau (*in-band network telemetry*).

**Résumé** : Le P4 a une structure de langage simple : il ressemble à du C allégé (ex. pas de pointeurs), auquel on a ajouté des mot-clefs natifs et des structures adaptées aux besoins des routeurs et commutateurs (entêtes, tables d'actions, déparseurs c-à-d sérialiseurs). Cela rend le p4 accessible aux ingénieurs réseaux n'ayant peu ou pas de d'expertise en semiconducteurs ou en logiciel. La facilité d'ajout de fonctionnalités leur donne la possibilité d'ajouter des fonctionnalités jusqu'alors hors d'atteinte, en particulier des mesures sur le réseau.

Les programmes P4 définissent le **plan de données**, séquence de blocs faite d'un parseur d'entêtes suivi tables d'actions, du déparseur et d'autres blocs définis en-dehors du langage (*externs*). Un P4 ne définit *pas*, par contre, de quelle manière les tables sont remplies. Cette fonction est laissée au logiciel de contrôle SDN externe nommé *plan de contrôle*. P4Runtime est l'un d'eux.

Le compilateur passe le programme vers une cible qui peut être n'importe quoi, d'un programme exécuté par un CPU, le commutateur simulé BMv2, du HDL pour un FPGA, ou des ASIC comme le Tofino. L'architecture P4 définit les contraintes de la cible, ce qui fait que les programmes P4 ne sont pas totalement portables.

**Avis** : Un premier aperçu prosaïque de ce qu'est le P4 et ses applications commerciales les plus évidentes.

## 2.4 What you should know about P4 programming language P4 programmable switch, [9], Afterfusion, Janvier 2022

**Pertinence :** 2 (article d'introduction au P4) **Sujets :** langage P4

**Problématique :** Quel est l'intérêt du P4, et de quoi est fait un commutateur P4 physiquement ?

**Prérequis :** Aucun

**Réponse :** Le P4 rend les commutateurs indépendants du protocole, permet aux ingénieurs réseau d'ajouter des applications maison dans leur équipement. Les commutateurs exécutent des fonctions du réseau normalement laissées aux OS des CPU : *load-balancing*, sécurité, pare-feu, fonctions infonuagiques (je vois pas trop lesquelles), diagnostics et télémétrie. Le Tofino possède pas mal le marché à lui tout seul.

**Résumé :** On peut ajouter des features au ASIC après installation et l'adapter à n'importe quel protocole. Il y a déjà des exemples d'applications commerciales réussies : Facebook a un *load-balancer* TCP sur le Tofino ; Stratum vend des commutateurs intelligents avec fonctionnalité nuagiques dessus : *load-balancer*, *flow control*, INT. La télémétrie est difficile voire impossible à mettre en place avec des équipements traditionnels, parce qu'une télémétrie implémentée en logicielle ne peut pas être aussi rapide que le trafic commuté. Les auteurs en profitent pour faire la promotion de leur commutateur intelligent, qui intègre un Tofino, un x86 Intel et des SoCs ARM maison appelés DPU pour accélérer les fonctionnalités réseau comme la télémétrie.

**Avis :** Un article commercial, mais qui a le mérite de montrer l'architecture d'un vrai matériel P4 fabriqué. Pour obtenir les features qu'ils voulaient, ils ont dû se rendre jusqu'à la fabrication d'un ASIC custom et payer la licence ARM dessus ; ce qui montre bien que le Tofino est limité dans ce qu'il peut faire. Aussi, leur commutateur est un gros investissement monétaire... est-ce-que les fonctionnalités réseau supplémentaire justifient un tel prix.

## 2.5 Getting started with P4, [10], Rijsman, 2019

**Pertinence :** 3 (tutoriel pour monter une VM P4 et envoyer quelques paquets tests) **Sujets :** langage P4, environnements de développement

**Problématique :** Quelles sont les étapes pour configurer un environnement P4 simple, monter un commutateur P4 basique, et le faire fonctionner sur le BMv2 ?

**Prérequis :** Maîtrise de VirtualBox/VMWare/autre [7] (l'auteur fait tout dans une boîte AWS, mais c'est quand même plus pratique avec une VM) ; protobuf ; interfaces virtuelles Linux

**Réponse :** Liste des étapes ci-dessous :

**Résumé :**

- Monter une VM Ubuntu 20.04.
- Ouvrir le SSH dessus (la manière la plus simple de le faire est de régler une IP statique sur la VM avec `netplan`, configurer le réseau de la VM en *bridge*).
- Installer le million de dépendances `apt` - sauf `texlive-full`, je ne vois pas pourquoi ils suggèrent ce paquet plus lourd à lui seul que tout les autres, ça fonctionne très bien sans.
- Compiler et installer protobuf
- Compiler et installer BMv2
- Taper le code P4 basé sur `<core.p4>` et `<v1model.p4>`.
- Compiler avec `p4c` vers le BMv2. Ils sort un `.json` lisible par le simulateur comportemental.
- Ajouter une paire d'interface virtuelles `veth` à la boîte Linux.
- Lancer le commutateur `simple_switch`, lui passer les interface virtuelles en option
- Lancer le plan de contrôle `simple_switch_CLI`, remplir la table de routage IP.
- Lancer des renifleurs de paquets sur les interfaces virtuelles, par exemple `tcpdump`.
- Envoyer des paquets test avec `scapy`. Selon l'IP de destination choisie ils apparaissent dans les fenêtres `tcpdump` en sortie.

**Avis :** Le seul tutoriel de référence trouvable qui explique comment monter l'environnement de A à Z, sans utiliser la VM préfabriquée du répo officiel. Il marche du premier coup. Par contre il n'explique pas *pourquoi* on a besoin de telle ou telle dépendance. Ça reste à explorer. Les outils sont déjà matures et fonctionne déjà bien. Reste à voir comment ça se passe quand on veut compiler vers des cibles matérielles.



## 2.6 An exhaustive survey on P4 programmable data plane switches : taxonomy, applications, challenges, and future trends, [11], Kfoury et al., Mai 2021

**Pertinence** : 3 (excellente revue des domaines d'applications des commutateurs programmables). Les sections VI-XII pas analysées dans le détail. **Sujets** : télémétrie, *load-balancing*, *in-network computing*, *middlebox functions*, IoT, sécurité, tests et vérification.

**Problématique** : Quel est l'état de l'art des commutateurs à plan de données programmables, et dans quelles directions vont-ils continuer à évoluer ?

**Prérequis** : Aucun

**Réponse** : Les équipements réseau numériques ont commencé dans un paradigme propriétaire, inflexible car de conception fermée. Le besoin d'interopérabilité et de standardisation a fait en sorte, dans ce paradigme, qu'il est extrêmement difficile de changer un protocole existant - la pile réseau s'« ossifie ». Dans les quelques dernières années on a flexibilisé le plan de contrôle avec SDN, mais le plan de données est resté fixe. L'étape suivante naturelle a été l'émergence de plan de données programmables, dont le P4 est devenu le langage standard de-facto. De nouvelles architectures de commutateurs en ont résultées. Les équipements propriétaires traditionnels vont être lentement mais surement remplacés par des équipements programmables.

### Table des matières :

- I Introduction
- II Related surveys
- III Traditional switches and SDN
- IV Programmable switches
- V Methodology and taxonomy
- VI Surveyed work
  - I In-band Network Telemetry
  - II Network performance
  - III Middlebox functions
  - IV In-network computing
  - V IoT
  - VI Cybersecurity and attacks
  - VII Network and P4 Testing
- VII Challenges and future trends
  - A Memory capacity (SRAM & TCAM)
  - B Resource accessibility

- C Arithmetic computations
- D Network cooperation
- E Control-plane intervention
- F Security
- G Interoperability
- H Programming simplicity
- I Deep programmability
- J Modularity and virtualization
- K Practical testing

## VIII Conclusion

### Résumé :

I - L'ossification des protocoles a eu lieu et est la conséquence directe du mainmise d'une poignée de fabricants d'équipement réseau sur le marché, utilisant des designs propriétaires non-reconfigurables. Le nombre de RFC sur les protocoles standards explose depuis les années 1990-2000, ce qui montre qu'il est difficile de faire changer un protocole pour soi sans forcer un changement au protocole sur tout le standard. Cela suggère qu'il faut changer de paradigme si l'on veut permettre à nouveau une innovation rapide sur les protocoles.

II - Jusqu'à ce jour il n'existait aucune revue exhaustive et extensive du langage P4. Les revues existantes ciblent des aspects spécifiques du domaine : Stubb sur les compilateurs et interpréteurs ; Darghali sur la sécurité. Cordeiro sur les fonctionnalités et possibilités du langage. Satapathy sur l'évolution depuis les équipements fixes jusqu'à la SDN, puis au plan de données programmable. Bifulco sur la taxonomie des commutateurs programmables. Kaljic sur la SDN en général. Kamman sur la SDN aussi, mais en touchant aussi à la télémétrie et au *in-network computing*. Tan sur l'évolution de la télémétrie pré-SDN, avec SDN, et avec le P4. Zhang sur le traitement *stateful*.

III - La SDN permet aux opérateurs de diriger le plan de contrôle depuis un point central. Le P4 étend la SDN, la rend indépendante de la cible ([en principe!]). En règle générale, le cours naturel des technologies se d'évoluer du matériel et des langages fixes à des équipements programmables et programmés dans des langages haut-niveau. PISA est une architecture générique de cible pour le P4, de la même manière que le x86 est une architecture générale pour les CPUs programmés en C.

IV - PISA est le gabarit d'architecture d'un commutateur générique. Il définit le modèle général d'un parseur comme machine à état, suivi d'une chaîne de tables d'actions, elles-mêmes composées de blocs de mémoires et de petits 'ALUs', le tout terminé par un sérialiseur programmable. Le  $P4_{16}$  a étendu le  $P4_{14}$  pour s'adapter à plus de cibles.

Avantages des commutateurs programmables : ils rendent l'adoption de nouvelles révisions plus rapide, renversent le flot de développement de *bottom-up* à *top-down*. Les opérateurs réseaux peuvent

créer de nouvelles fonctionnalités pour obtenir un avantage compétitif sans avoir à le partager (ou le faire adopter dans un standard). Contrairement à ce à quoi on pourrait s'attendre, les équipements programmables améliorent la performance parce qu'ils peuvent ne contenir que les fonctionnalités dont on a besoin ; là où les ASICs fixes doivent incorporer tout un tas de fonctionnalités secondaires, dont la grande majorité ne sera jamais utilisées, juste au cas-où.

L'écart de performances entre les puces de commutations et la commutation par CPU l'élargit de plus en plus :  $\times 5 - 10$  en 1990-2000,  $\times 100$  maintenant. Cela s'explique par les multiples niveaux de parallélisme exploitables dans un ASIC/puce programmable : pipelinage entre l'*ingress* et l'*egress*, traitement de plusieurs entêtes en parallèle. En fait, le traitement parallèle de plusieurs champs en même temps fait que le commutateur se comporte comme un processeur VLIW, où l'entête du paquet est comme une longue instruction.

V - L'immense majorité (90 %) des publications sur le P4 utilise soit des simulateurs comportementaux ([j'imagine BMv2]), soit des ASICs. Le 10% restant fonctionne avec des smartNIC ou le NetFPGA.

Les sections VI-XII passent en revue toutes les applications possibles de la recherche dans le domaine.

VI - La INT est standardisée et spécifiée dans un document rédigé par Barefoot et d'autre entreprises du secteur. La INT donne beaucoup plus de visibilité sur le réseau que les outils du style `ping` et `traceroute`.

Fonctionnement standard de la télémétrie : un premier terminal étiqueté 'source INT' insère une entête 'instruction INT' n'importe où entre les entêtes existantes (pour ça reste flexible et indépendant du protocole). Celle-ci indique aux prochains routeurs traversés quelles mesures prendre. À chaque saut l'équipement ajoute une entête contenant les données récoltées d'après l'instruction. En bout de chaîne, le dernier saut enlève toutes les entêtes INT du packet et les poussent vers un noeud 'INT collector' qui en fait l'agrégation. Le reste du paquet est transmis à sa destination normalement, tout le processus de INT est transparent du point de vue de celle-ci.

La recherche actuelle en INT essaie de réduire son coût d'*overhead* et de simplifier le traitement des données au collecteur.

VII - Les principaux problèmes de performance traités par la recherche P4 sont le contrôle de congestion et la gestion des file (AQM, Active Queue Management), les mesures de performance et les diagnostics.

La plupart des mesures marchent avec des protocoles basées sur des requêtes, mais pour l'instant rien n'est standardisé/optimisé. La CC (contrôle de congestion) et la QoS (les deux sont liés) s'implémentent en séparant le trafic entre les équipements et en ralentissant le trafic à la source (*throttling sender traffic*), le tout en exploitant les données de télémétrie et l'inspection jusqu'à la couche application.

VIII - Définition de *middlebox* : équipement réseau utilisé pour réaliser n'importe quelle fonction autre que la commutation/routage classique, ex. NAT, conversion 4à6, 6à4, pare-feu.

Fonctions les plus étudiées en P4 : équilibrage des charges (demande du *stateful processing*) ; les caches sur le réseau (*in-network caches*) peuvent fonctionner mais demandent encore plus de travail : compression, choix des données à mettre en cache, minimisation de l'impact sur les performances globales ; décharge de fonctions télécom (?), utiliser directement les commutateurs comme *broker* de protocoles comme MQTT.

IX - Principaux candidats pour accélération-sur-réseau (*in-network acceleration*) : détermination du consensus dans les algorithmes par vote, phase d'aggrégation dans l'apprentissage machine. Pour le moment toutes ces pistes sont étudiées mais on ne fait tourner que des versions simplifiées des protocoles en question.

X - Les commutateurs intelligents peuvent agréger les paquets de plusieurs appareils IoT en un seul avant de l'expédier. L'intérêt de l'aggrégation vient de la petite taille des données typiquement émises par ces appareils : les entêtes représentent une grosse part du coût des données. Le paquet agrégé n'a qu'une seule entête et économise donc une partie des coûts. Mais ça se fait au prix qu'une latence allongée, puisqu'on attend que plusieurs paquets arrivent avant de les expédier.

XI - Les comm. int. peuvent agir sur la sécurité à toutes les couches du réseau.

Principaux usage : détection des *heavy-hitters* (client utilisant une bande-passante trop large ou ayant un nombre anormalement élevé de connexions ouvertes) distribuée sur plusieurs commutateurs ; sans télémétrie P4 on ne peut pas les détecter si leur trafic anormal est distribué entre plusieurs commutateurs/routeurs.

Autres fonctions de sécurité étudiées : implémentation de cryptographie sur le commutateur (pour le moment laissée aux terminaux), obfuscation de la topologie, cacher les entêtes de source/destination pour améliorer la vie privée ; contremesures : détection DDoS, pare-feus à chaque couche (pas juste L3-L4), filtrage URL, filtrage de mot-clef.

XII - Les méthodes de débogage réseau actuelles ne sont pas adaptées au matériel reconfigurable, qui offre beaucoup plus de services peu ou pas standardisés. Les possibilités ajoutées sont autant d'ouverture par les lesquels on introduit des bugs, susceptibles de perturber le réseau. Les travaux futurs pourraient travailler à ajouter de la résilience au réseau, en prévision : auto-diagnostics, auto-réparation à partir des données de télémétrie.

XIII - Défis et travaux futurs :

A - Limites sur la taille des mémoires : certaines applications, entre autres les caches-sur-réseau, font que l'on stocke de plus en plus de données dans de grandes tables.  $\Rightarrow$  Solution : utiliser des mémoires *off-chip*, mais ça vient avec d'autres limites : bande passante, latence, etc.

B - Les implémentations P4 actuelles n'ont de mécanisme de bassin de ressources (*pooling*),

par exemple la mémoire des tables n'est pas partagée, et donc inefficacement utilisée.  $\Rightarrow$  Solution possible : avoir un bassin de mémoire centralisé et accessible depuis une Xbar ? Idem avec des coeurs de processeurs.

C - Les calculs sur réseau sont limités par les contraintes de délais dans les nanosecondes, d'où la limite de leur complexité matérielle. Il n'est pas envisageable, par exemple, de faire du calcul à point flottant. Les applications implémentables sont donc limitées : difficile de faire de l'AQM, de la gestion de trafic, ou de l'apprentissage machine (qui implique généralement des points flottants).  $\Rightarrow$  Utiliser des approximations de fonctions avec pertes de précision, ou alors laisser le plan de contrôle précalculer des valeurs et stocker les résultats dans des tables.

D - Coopération sur la réseau : l'état du réseau doit être synchronisé entre les commutateurs, par exemple pour la gestion de trafic ou la détection DDoS  $\Rightarrow$  Il existe des protocoles comme P4Sync pour que les équipements puissent échanger des messages de contrôle entre eux.

E - Les interactions avec le plan de contrôle ralentissent le routage, ex. si un paquet ne correspond à aucune des entrées actuelles de la table. L'article ne suggère pas de solution concrètes pour minimiser les appels au plan de contrôle.

F - En réponse à tous les problèmes de sécurité, les commutateurs devraient inspecter le trafic pour y repérer les patterns correspondant à des attaques.

G - Interopérabilité : les anciens équipements ne seront pas remplacés d'un coup par des équipements P4, il faut organiser la coexistence des deux pour un remplacement incrémental. L'une des possibilités sera de monter des commutateurs P4 par-dessus les lignes existantes pour écouter le trafic et collecter des données de télémétrie sans perturber le réseau existant.

H - La programmabilité a bien des avantages, mais amène avec lui les difficultés intrinsèques du logiciel : les bogues inévitables et peuvent rester en production longtemps ; la complexité mal gérée, l'élargissement de l'éventail de compétence demandées des opérateurs réseau.  $\Rightarrow$  Solution possible : des outils de développement plus visuels  $\Rightarrow$  Le compilateur P4All et son extension de langage optimisent l'utilisation des ressources au lieu de la laisser aux programmeurs forcément imparfaits.

I - Dans un avenir plus ou moins lointain on peut envisager un réseau « programmable en profondeur », dans lequel tout les éléments du réseau (Serveurs, NICs, routeurs) sont programmables et capables de s'ajuster en fonction de la congestion du réseau en utilisant les données de télémétrie (système en boucle ouverte).

J - Reconfiguration à chaud : à l'heure actuelle les équipements sont mis hors-ligne pendant qu'ils sont reconfigurés ex. avec un nouveau programme P4. Il existe plusieurs frameworks implémentant la reconfiguration à chaud, mais tous ont un coût en performances.

K - La plupart des tests et vérifications sont faits en simulations/émulations de réseaux à petite

échelle, parce qu'une simulation sur CPU ne peut pas atteindre le niveau de performances d'un réseau réel à grande échelle et à pleine charge. Il faut donc trouver un moyen de faire des tests physiques.  $\Rightarrow$  Solutions possible : TurboNet, un émulateur performant. P4Campus, une méthode pour chercheurs visant à tester une implémentation à l'échelle du réseau d'un campus universitaire.

L - L'opération du réseau par des humains se prête à l'erreur. Dans un futur distant on peut imaginer automatiser au complet le contrôle du réseau, avec un contrôle en boucle-fermée exploitant la télémétrie.

**Avis :** Un tour d'horizon exhaustif et très bien écrit de toutes les possibilités des équipements programmables. C'est un point de départ utile pour trouver un sujet de recherche. Seul point manquant : l'aspect matériel n'est presque pas abordé ; j'aurais aimé voir un commentaire sur les plateformes existantes en production et en prototypage.

### 3 Liste des références

#### Références

- [1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese et D. Walker, “P4 : programming protocol-independent packet processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, n°. 3, p. 87–95, juill. 2014. [En ligne]. Disponible : <https://dl.acm.org/doi/10.1145/2656877.2656890>
- [2] “MicroNugget : What is Multi-Protocol Label Switching (MPLS) ? - YouTube.” [En ligne]. Disponible : <https://www.youtube.com/watch?v=huKkCK8AJ7I>
- [3] “Multiprotocol Label Switching,” févr. 2023, page Version ID : 1137499050. [En ligne]. Disponible : [https://en.wikipedia.org/w/index.php?title=Multiprotocol\\_Label\\_Switching&oldid=1137499050](https://en.wikipedia.org/w/index.php?title=Multiprotocol_Label_Switching&oldid=1137499050)
- [4] “VLAN,” janv. 2023, page Version ID : 1134281103. [En ligne]. Disponible : <https://en.wikipedia.org/w/index.php?title=VLAN&oldid=1134281103>
- [5] “IEEE 802.1Q,” déc. 2022, page Version ID : 1128288082. [En ligne]. Disponible : [https://en.wikipedia.org/w/index.php?title=IEEE\\_802.1Q&oldid=1128288082](https://en.wikipedia.org/w/index.php?title=IEEE_802.1Q&oldid=1128288082)
- [6] “EtherType,” mars 2022, page Version ID : 192325643. [En ligne]. Disponible : <https://fr.wikipedia.org/w/index.php?title=EtherType&oldid=192325643>
- [7] *Oracle VM VirtualBox User Manual*, 2004. [En ligne]. Disponible : <https://www.virtualbox.org/manual/>
- [8] P. Parol, “P4 Network Programming Language—what is it all about ?” avr. 2020. [En ligne]. Disponible : <https://codilime.com/blog/p4-network-programming-language-what-is-it-all-about/>
- [9] “What you should know about P4 programming language& P4 programmable switch,” janv. 2022. [En ligne]. Disponible : <https://cloudswit.ch/blogs/what-you-should-know-about-p4-programming-language-p4-switch/>
- [10] B. Rijsman, “Getting Started with P4,” oct. 2019. [En ligne]. Disponible : <https://opennetworking.org/news-and-events/blog/getting-started-with-p4/>
- [11] E. F. Kfoury, J. Crichigno et E. Bou-Harb, “An Exhaustive Survey on P4 Programmable Data Plane Switches : Taxonomy, Applications, Challenges, and Future Trends,” *IEEE Access*, vol. 9, p. 87 094–87 155, 2021. [En ligne]. Disponible : <https://ieeexplore.ieee.org/document/9447791/>