Titre: Title:	Des milliards de flux : l'analyse d'applications à gestion d'états dans le contexte des commutateurs programmables
Auteur: Author:	Vincent Pascal Brouillard
Date:	2023
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Brouillard, V. P. (2023). Des milliards de flux : l'analyse d'applications à gestion d'états dans le contexte des commutateurs programmables [Master's thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/53428/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/53428/
Directeurs de recherche: Advisors:	J. M. Pierre Langlois, & Yvon Savaria
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Des milliards de flux: l'analyse d'applications à gestion d'états dans le contexte des commutateurs programmables

VINCENT PASCAL BROUILLARD
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Mai 2023

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Des milliards de flux:
l'analyse d'applications à gestion d'états
dans le contexte des commutateurs programmables

présenté par **Vincent Pascal BROUILLARD**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Tarek OULD-BACHIR, président
Pierre LANGLOIS, membre et directeur de recherche
Yvon SAVARIA, membre et codirecteur de recherche
Jean Pierre DAVID, membre

DÉDICACE

À ma mère,
qui m'a exposé à l'ingénierie.

REMERCIEMENTS

Je tiens à remercier toutes les personnes ayant contribué à l'accomplissement de ce mémoire.

Je voudrais remercier dans un premier temps mes directeurs, Pierre LANGLOIS, professeur en génie informatique ainsi que Yvon SAVARIA, professeur en génie électrique. Leur disponibilité, leur écoute et leur soutien constants ont été essentiels pour la rédaction de ce mémoire.

Je remercie également tous les membres du jury pour leurs commentaires qui ont permis d'améliorer ce mémoire.

Je tiens à souligner le soutien des partenaires industriels chez Intel, Kaloom et Noviflow sans lequel ce mémoire n'existerait pas.

Je tiens à témoigner toute ma reconnaissance aux personnes suivantes qui ont permis de pousser ma réflexion :

André BÉLIVEAU de Noviflow, qui m'a permis d'identifier les protocoles et applications à gestion d'états.

Bochra BOUGHZALA et Ludovic BÉLIVEAU de Kaloom, qui m'ont accompagné dans mon exploration des réseaux 5G.

Je remercie également toute l'équipe du département de génie informatique qui m'a permis de me rendre à ce point-ci de mes études.

Un gros merci à Pedro CYBIS et Jacques BROUILLARD qui ont supporté mon moral pour toute la durée de ma maîtrise.

Le soutien pour la capture internet du CAIDA provient de la *National Science Foundation*, du *US Department of Homeland Security* et des membres du CAIDA.

Support for CAIDA's Internet Traces is provided by the National Science Foundation, the US Department of Homeland Security, and CAIDA Members.

RÉSUMÉ

Les réseaux, et spécifiquement l'internet, ont transformé radicalement notre société. Or, l'omniprésence d'objets connectés, la proéminence de l'infonuagique et même les pandémies mettent les réseaux à rude épreuve. Les réseaux ont plus que jamais besoin de s'adapter finement à l'utilisation que l'on en fait. Les commutateurs programmables viennent permettre de bénéficier de la performance du matériel avec la flexibilité du logiciel. Les applications avec gestion d'états permettent d'adapter le traitement en fonction d'événements passés. La granularité de ce genre de traitement pose toutefois problème. Afin de pouvoir effectuer la gestion d'états au moyen de ces commutateurs, nous discutons des contraintes propres à ce type d'application. Nous présentons la littérature entourant les commutateurs programmables. Nous illustrons ensuite les cas de six applications avec gestion d'états. Parmi ces applications, le protocole TCP fait l'objet d'une analyse plus approfondie. Nous introduisons une approche de traitement des flux longs à partir de trace volumineuse ainsi qu'une approche de traitement hiérarchique des états. Deux programmes ont été conçus pour la réalisation d'une telle analyse. Ce mémoire porte sur l'analyse d'applications avec gestion des états et les contraintes qui leur sont propres dans le contexte des commutateurs programmables.

ABSTRACT

Networks, and specifically the Internet, have radically transformed our society. However, the omnipresence of connected objects, the prominence of cloud computing and even pandemics put networks to the test. More than ever, networks need to adapt finely to the use that we make of them. Programmable switches allow you to benefit from the performance of hardware with the flexibility of software. Applications with state management allow processing to be adapted according to past events. However, the granularity of this type of processing is problematic. In order to be able to perform state management by means of these switches, we discuss the constraints specific to this type of application. We present the literature surrounding programmable switches. We then illustrate the cases of six applications with state management. Among these applications, the TCP protocol is the subject of further analysis. We introduce a long flow processing approach for large network capture as well as a hierarchical state processing approach. Two programs have been produce to help conduct such an analysis. This thesis focuses on the analysis of applications with state management and the constraints specific to them in the context of programmable switches.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ASIC	Application-Specific Integrated Circuit
BCAM	Binary Content Addressable Memories
CAM	Content Addressable Memories
DAG	Directed Acyclic Graph
eFSM	Extended Finite State Machine
FPGA	Field-Programmable Gate Array
GPRS	General Packet Radio Service
GTP	GPRS Tunneling Protocol
GSN	GPRS support nodes
GSM	Global System for Mobile Communications
IETF	Internet Engineering Task Force
NPU	Network Processing Unit
OSI	Open Systems Interconnection
PIFO	Push In, First Out
PISA	Protocol-Independant Switch Architecture
RAM	Random Access Memory
RDMA	Remote Direct Memory Access
RMT	Reconfigurable Match-action Table
TCAM	Ternary Content Addressable Memories
TCP	Transport Control Protocol
VLIW	Very Long Instruction Word

CHAPITRE 1 INTRODUCTION

1.1 Contexte du mémoire

Les réseaux, et spécifiquement l'internet, ont transformé radicalement notre société. À titre d'exemple, en cas de crise lors d'une pandémie, la nécessité des réseaux est flagrante. Que ce soit pour le travail à distance, pour le divertissement ou pour obtenir de l'information fiable, l'internet est un incontournable. Or, l'omniprésence d'objets connectés, la proéminence de l'infonuagique et même les pandémies mettent les réseaux à rude épreuve. Les réseaux ont plus que jamais besoin de s'adapter finement à l'utilisation que l'on en fait. Citons à titre d'exemple les réseaux 5G qui offrent une variété de modes de transmission pour différents besoins avec des contraintes en apparence incompatibles.

Cette évolution vers la flexibilité n'est pas qu'une tendance. La séparation des plans réseaux en plan de contrôle et plan de données a ouvert la voie à une séparation des besoins du trafic réseau. Les réseaux définis par logiciel ont permis de profiter de cette séparation afin d'améliorer la capacité d'adaptation et de fournir des outils d'automatisation. Toutefois, si une approche logicielle suffit pour le plan de contrôle, les besoins en matière de performance du plan de données nécessitent souvent du matériel dédié. Les commutateurs programmables viennent permettre de bénéficier de la performance du matériel avec la flexibilité du logiciel. Ces commutateurs offrent une grande flexibilité aux opérateurs, leur permettant de s'adapter aux différents événements du réseau.

Les applications avec gestion d'états forment une classe d'applications qui se souviennent d'événements passés et utilisent l'information pour modifier le traitement des données. La gestion des états signifie une adaptation très fine aux événements du réseau, mais au prix de l'utilisation de la mémoire. Ce type d'applications se base sur l'identification de groupes de paquets reliés logiquement autour d'une application. Ce groupe de paquets, aussi appelé flux et, dans certains cas, session, évolue de manière indépendante des autres groupes. Cela nécessite de pouvoir identifier l'appartenance au groupe, d'effectuer le traitement selon l'état courant puis de mettre à jour l'état. La gestion des états pousse donc la capacité d'adaptation et la flexibilité vers une granularité fine. Les commutateurs programmables semblent donc offrir un bon compromis entre une approche matérielle et une approche logicielle pour cette classe d'applications. Il est donc pertinent de s'intéresser aux caractéristiques propres à ce type d'applications.

1.2 Problème considéré

Ce mémoire introduit quelques concepts entourant la gestion d'états et les commutateurs programmables. Il contient également une analyse de différents protocoles et applications avec gestion d'états d'un point de vue axé sur les commutateurs programmables. L'un de ces protocoles, TCP, bénéficie d'une analyse approfondie. Cette analyse porte sur le comportement des flux, ce qui nécessite une méthodologie particulière en raison du nombre de ces flux. En effet, pour avoir une bonne vision du comportement des flux, cela implique de comptabiliser plusieurs métriques par flux. Ces métriques permettent d'estimer l'ampleur du défi de la gestion d'états en matériel. Deux programmes ont été produits pour obtenir ces métriques. Ces outils d'analyse et la discussion entourant la gestion d'états pour les commutateurs programmables constituent l'apport principal de ce mémoire.

La gestion des états est un défi pour les commutateurs. La quantité d'informations à retenir, la durée de rétention et la fréquence de mise à jour ont chacun un impact majeur sur la capacité de traitement. Trouver le compromis optimal entre le temps d'accès et la quantité de stockage n'est pas chose facile. Ce n'est pas un nouveau problème, et l'utilisation de la hiérarchie de mémoire dans les processeurs continue de faire ses preuves [10]. Par contre, ce compromis est dépendant du domaine d'application. Dans un même commutateur, plusieurs applications distinctes avec des besoin variés coexistent. Que ce soit des contraintes de faible latence pour un média en temps réel, des contraintes de haut débit ou les deux simultanément, différencier les services permet de satisfaire tout le monde. La complexité du traitement a longtemps nécessité une approche logicielle pour ce type d'applications. Toutefois, l'arrivée des commutateurs programmables et de l'architecture PISA ouvre la voie vers une approche plus performante. Ces commutateurs allient la flexibilité des languages spécifiques au domaine des réseaux et la performance d'un processeur conçu pour le traitement réseau.

1.3 Objectifs de recherche

Étant donné la problématique énoncée dans les sections précédentes, nous considérons les objectifs de recherche suivants dans ce mémoire :

— Analyser des applications et des protocoles avec gestion des états pour des réseaux informatiques à haut débit.
— Proposer une méthodologie d'analyse des flux dans une trace de réseau volumineuse afin d'analyser le comportement des flux de paquets du protocole TCP.

En terme d'analyse de flux, on tente de déterminer les contraintes de mémoire pour implémenter la gestion de flux : quantité, fréquence de mise à jour, et persistance des données.

1.4 Approche suivie

Afin d'atteindre ces objectifs, nous analyserons dans ce mémoire les différents concepts entourant la gestion des états. Nous considérerons les architectures configurables de type PISA comme étant la plateforme pour ces applications. La gestion des états étant un sujet vaste, nous nous bornerons à quelques applications typiques. Toutefois, le manque de données d'utilisation sur ces applications rend l'analyse des contraintes plus conceptuelle. Ce n'est toutefois pas le cas du protocole TCP, qui a fait l'objet d'une analyse poussée et pour lequel bon nombre de captures de trafic en différents points du réseau existent. Il est important de souligner que l'objectif ici est d'utiliser une capture de trafic afin de mieux comprendre les contraintes des applications à gestion des états en prenant TCP comme exemple. TCP est un protocole mature qui sert de modèle pour le comportement des flux. En effet, TCP est un protocole de transport assez bas dans le modèle OSI, et il sert de base pour nombre d'autres protocoles et applications. Ainsi, l'analyse approfondie de TCP présentée dans ce mémoire peut être utile pour la conception d'applications à gestion des états. Cette analyse est orientée sur le comportement des flux, en particulier les flux de longue durée. Leur longue durée implique une présence en mémoire prolongée. Les flux de courte durée ne comportent en général pas assez de paquets transigés pour qu'une gestion des états puisse en affecter significativement le traitement.

1.5 Plan du mémoire

Le mémoire est divisé en quatre parties. D'abord, l'introduction offre un contexte sur ce qui a été réalisé. Ensuite, une revue de littérature couvre les fondements des commutateurs programmables ainsi que les différents concepts du réseau qui les entoure. Cette revue couvre également les différentes approches d'optimisation de la gestion des états en utilisant des commutateurs programmables. Le troisième chapitre part sur la base de la revue de littérature pour couvrir différentes applications avec gestion des états. Le dernier chapitre présente une méthodologie de traitement de captures volumineuse. Grâce à cette méthode, diverses métriques sur le comportement des flux TCP sont présentées et sommairement discutées.

CHAPITRE 2 REVUE DE LITTÉRATURE

La multiplication des objets connectés et la proéminence de l'infonuagique imposent de grandes contraintes sur les réseaux traditionnels [11]. Les appareils mobiles et les machines virtuelles font en sorte que les réseaux doivent être de plus en plus flexibles. Pour rendre l'infrastructure plus flexible, elle a d'abord été scindée en un plan de contrôle et un plan de donnée [12]. Cela a permis de mieux contrôler la configuration des réseaux. Ensuite, pour s'affranchir de l'aspect fixe des réseaux physiques, une multitude de nouveaux protocoles ont permis de virtualiser les connexions. Les protocoles comme VXLAN, NVGRE et Geneve ont permis de créer des réseaux arbitraires au-dessus du réseau physique [13].

2.1 Séparation des plans réseau

Afin de rendre les réseaux à la fois flexibles et performants, au tournant des années 2000, un nouveau paradigme réseau s'est formé [12] [13]. Pour satisfaire les besoins grandissants de l'internet, le réseau a été scindé en deux plans conceptuels. D'une part, le trafic des données couvre le transfert de l'information et l'utilisation générale du réseau. D'autre part, le trafic de contrôle permet d'organiser le trafic de données et de modifier l'état du réseau. Normalement, on considère le plan de données comme étant le trajet rapide (haut débit), tandis que le plan de contrôle est le trajet lent (faible débit). Cette distinction provient de la différence en matière de volume de trafic.

Plan de contrôle. Pour que le réseau fonctionne convenablement, un ensemble de données d'état du réseau est partagé à chaque instant. Par exemple, pour permettre d'acheminer un paquet il faut que chaque commutateur puisse connaitre une portion du trajet et donc, qu'il doive avoir une connaissance de son voisinage réseau. Des protocoles comme RIP, OSPF et BGP permettent de transférer l'information des connexions de routeur et ainsi d'établir les différentes routes qu'un paquet peut employer. De plus, les différents protocoles de transfert de données n'ont pas tous la même tolérance à la latence. Par exemple, le transfert de vidéos souffre de la moindre variation de latence tandis que le transfert de textes (c.-à-d., HTTP) peut supporter de grandes variations de latence. Des protocoles de gestion de la qualité des services (QoS, *Quality of services*) permettent d'assurer le partage de l'état du réseau et ainsi assurer de respecter les contraintes propres à chaque service.

Plan de données. La majeure partie des données transitant sur le réseau se trouve dans le plan des données. Ce plan repose sur les tables de routage et les configurations des commutateurs modifiés dans le plan de contrôle. Les paquets sont donc traités de manière déterminée. De cette façon, il est possible d'optimiser le traitement au moyen de composants matériels. Ainsi, les communications peuvent avoir une latence réduite et un débit garanti. Par ailleurs, les connexions sont séparées en deux types : les connexions avec états (*Statefull*) et sans états (*Stateless*). Les états permettent de modifier le comportement du traitement pour les paquets futurs. C'est pour cette raison que l'on parle aussi de flux de paquets. Les paquets d'un flux sont regroupés de façon logique pour être traités de la même manière. Cela permet d'accélérer le traitement des paquets en réduisant le temps de traitement nécessaire.

2.2 Indépendance des protocoles

Les nouveaux besoins réseau de l'infonuagique et des systèmes distribués ont amené de nouveaux besoins en matière de protocoles. C'est pour cela que des normes comme VXLAN, NVGRE et Geneve ont été créées [13]. Toutefois, la capacité à traiter ces nouveaux protocoles demande de l'équipement réseau adapté, ce qui ralentit leurs développements et leurs supports. Cela a motivé la conception de composants réseau indépendants de protocoles où l'on peut décrire le protocole que l'on veut prendre en charge. Ainsi, la prise en charge d'un nouveau protocole se réduit à l'ajout d'une description du traitement nécessaire pour ce protocole.

Cette programmation des commutateurs nous offre de nombreux avantages. Par exemple, il serait possible d'ajouter de nouveaux protocoles, de retirer le support de protocoles ou d'ajouter et de retirer des fonctionnalités à volonté. L'ajout de nouveaux protocoles permet à la fois de supporter de nouveaux standards et de développer des protocoles «maison» pour des besoins internes à une entreprise. Le retrait ou la simplification de la prise en charge d'un protocole pourrait permettre de libérer des ressources qui seraient potentiellement inutilisées pour accélérer le transfert. De plus, il serait possible de profiter de la hiérarchie réseau pour centrer aisément le support de fonctionnalités là où elles sont vraiment nécessaires. On peut ainsi imaginer que le traitement de certains protocoles soit laissé à certains niveaux du réseau de la même manière que la couche réseau était laissée aux routeurs il y a une vingtaine d'années.

Par ailleurs, il serait possible d'ajouter de la télémétrie à même les paquets du plan de données. La compréhension du comportement de systèmes répartis en serait grandement amélio-

rée de même que le suivi des performances des services. En cas de panne, ce genre d'outils diagnostiques serait extrêmement pratique pour les gestionnaires de centre de données et les fournisseurs d'accès internet.

2.3 Éléments d'un commutateur

Afin de discuter de commutateur programmable, il est important de définir les fonctionnalités nécessaires. Toutefois, avant de présenter les aspects configurables d'un commutateur programmable il faut présenter les éléments communs à tous les commutateurs. Le traitement d'un commutateur procède minimalement en 3 étapes. On commence par l'analyse syntaxique qui nous fournit les champs d'entête et les protocoles. On effectue ensuite la classification des paquets et la recherche de règle appropriée. À partir des règles, il est possible de déterminer les opérations de traitement sur les entêtes pour préparer le paquet à être transféré vers sa destination. C'est dans cet ordre que les différents éléments sont abordés.

2.3.1 Analyse syntaxique

L'analyseur syntaxique permet d'identifier la structure d'un paquet entrant et d'en extraire les différents champs [14]. Ensuite, les différents champs sont transmis aux autres sections du commutateur pour être traités et transmis vers la destination du paquet.

Pour extraire les en-têtes, il faut établir la structure de l'encapsulation du paquet, c.-à-d. les sections de trame. Pour ce faire, il est possible de dresser un graphe qui permet de déterminer les successions possibles de protocoles. Par exemple, si l'on regarde la figure 2.1, un en-tête Ethernet peut être suivi d'un en-tête IPv4 ou IPv6 puis d'un en-tête TCP ou UDP. Chaque protocole possède son propre standard décrivant la signification et la taille de chaque portion d'en-tête. Il y a aussi une portion des champs qui se trouvent à la fin du paquet, ces champs servant surtout pour les sommes de contrôle. L'analyseur syntaxique doit pouvoir extraire les différentes valeurs après les avoir identifiées.

Un défi pour la conception d'un analyseur se trouve dans la longueur des champs. En effet, certains champs sont de taille variable ce qui rend plus complexe la création de circuits d'analyse. Cela impose de pouvoir gérer des décalages et d'avoir des mémoires et des ports d'accès adaptés en conséquence.

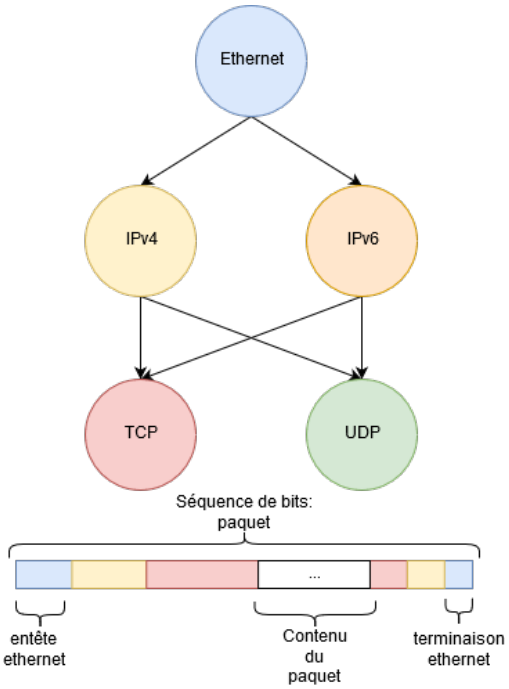


Figure 2.1 Graphe de dépendance de protocole et structure d'encapsulation d'un paquet.

En considérant que l'analyseur syntaxique se base sur les standards des différents protocoles, il est évident que l'indépendance de protocole peut rajouter une certaine complexité. En effet, dans ce cas, il faut fournir à l'analyseur la description des protocoles et le graphe de leur succession possible. L'analyseur doit ensuite pouvoir adapter ses composantes en fonction de cette configuration.

2.3.2 Classification et recherche

Dans un commutateur classique, la classification et l'analyse syntaxique sont souvent regroupées [14]. Toutefois, pour le cas d'un commutateur programmable et indépendant des protocoles, ces fonctionnalités sont scindées.

La classification permet d'analyser les différents paquets selon des règles établies à la configuration pour définir les traitements qui y seront appliqués. Par exemple, si un paquet Ethernet possède une étiquette correspondant au VLAN, que l'adresse de destination ne se trouve pas dans ce VLAN et qu'il n'y a pas de règle pour la communication entre les VLAN, alors le paquet ne sera pas transmis. Les paquets similaires sont alors rassemblés en flux (*flow*) et subissent les mêmes opérations. La classification est soit dépendante de l'état (*stateful*) soit

indépendante (*stateless*). Ainsi, le commutateur qui reçoit le premier paquet d'un flux peut modifier son état et alors modifier le traitement des paquets subséquents. Cette état doit toutefois être préservé pour la durée du flux. Il faut alors faire attention à la mémoire utilisé pour préservé cet état puisqu'elle croit avec le nombre de flux.

Lorsque la classification est plus complexe que la comparaison de deux valeurs, il est souvent nécessaire de faire une recherche parmi des tables de correspondance. Par exemple, si l'on doit transférer un paquet d'un réseau à un autre, il faut que les adresses se trouvent dans la table de routage. Pour vérifier si l'adresse se trouve dans cette table il faut des structures de recherche adaptées. Les *Content Addressable Memories (*ou CAM) sont des circuits pour effectuer des recherches en matériel et ainsi trouver l'adresse de l'espace mémoire qui correspond à la valeur que l'on cherche [14]. Cette adresse peut ensuite servir à obtenir une valeur dans un espace mémoire RAM. La valeur ainsi trouvée peut servir pour diverses opérations de traitement. Les CAM et RAM peuvent être modifiées par la configuration du commutateur. Il existe des CAM pour obtenir des correspondances exactes, nommées *binary CAM* ou BCAM. Il est aussi possible d'effectuer des correspondances partielles avec ce qu'on appelle des *Ternary CAM* ou TCAM. Ces deux composantes sont, avec les RAM, les composantes de base de la classification et de la recherche [14].

2.3.3 Opérations sur les paquets

Les opérations sur les paquets sont normalement déterminées par leur direction entrante (*ingress*) ou sortante (*egress*) [14]. Toutefois, la démarcation entre ces deux orientations n'est plus aussi claire de nos jours [14]. Les opérations d'entrée sont typiquement parmi les tâches suivantes (tirées de [14] p. 187) :

— Détection d'erreur.
— Vérification de sécurité et décryptage.
— Gestion du trafic (mesure et gestion de règle).
— Manipulation d'en-tête.
— Réassemblage de paquets.
— Priorisation et mise en file.
— Réacheminement de paquets.

Les opérations de sortie sont parmi les tâches suivantes (tirées de [14] p. 188) :

— Calcul et ajout des codes de correction d'erreur.
— Réacheminement de paquets.
— Segmentation et fragmentation.
— Gestion de trafic (mise en forme du trafic, synchronisation et ordonnancement).
— Priorisation et mise en file.
— Traitement de sécurité (c.-à-d., chiffrement).

Toutes ces manipulations dépendent grandement de l'analyse syntaxique et de la classification. En effet, les opérations requises sont déterminées par les protocoles et souvent par des règles de traitement définies lors de la configuration. Plusieurs opérations impliquent de manipuler plusieurs paquets en même temps, comme le réassemblage.

De plus, le traitement des paquets doit pouvoir rester cohérent malgré une certaine parallélisation du traitement. Cela signifie d'avoir de la mémoire partagée entre les différentes unités de traitement. Par ailleurs, entre les phases d'entrée et de sortie, il est souvent nécessaire d'avoir une zone tampon composée de files à priorité qui permettent d'ordonnancer le traitement des paquets.

2.4 Langages spécifiques aux domaines d'application

Dans le but de s'adapter à une nouvelle réalité on a, parfois, recourt à un nouveau champ lexical, mieux adapté. Pour la programmation, lorsqu'on est confronté à un nouveau domaine, il faut parfois créer un nouveau langage plus approprié. Ce langage permettant d'exploiter un niveau d'abstraction plus élevé, on peut alors simplifier les programmes et le travail des programmeurs. Dans le cas des réseaux programmables, cela permet d'apporter des modifications au comportement des commutateurs en leur fournissant une spécification du comportement attendu [13]. De plus, cette description permet de faire abstraction du matériel et ainsi elle peut être portée sur différentes architectures de différents fabricants. Un programmeur ou un opérateur réseau n'aurait qu'à connaitre le langage sans connaitre les détails de l'implémentation du composant réseau. En effet, on délègue au compilateur la responsabilité d'adapter le programme à l'architecture visée. Cela implique que le compilateur connaisse l'architecture sous-jacente ainsi que les ressources disponibles. À la différence d'un langage de programmation standard, un langage spécifique à un domaine permet de mieux épouser les contraintes du domaine et d'adapter le modèle de programmation en conséquence. Dans le cas des réseaux, la principale contrainte est la latence de traitement. Étant donné que l'on

souhaite traiter un grand nombre de paquets par seconde, il est nécessaire que le délai de traitement soit le plus court possible.

Pour augmenter le débit de traitement, la solution la plus fréquemment utilisée est la concurrence. Or, il ne s'agit pas d'exécuter plusieurs fois la même opération sur tous les paquets, mais bien des opérations propres à chaque type de paquets. Cette complexité est normalement reléguée au modèle abstrait de l'architecture et donc le langage permet d'en simplifier la gestion. Le modèle d'abstraction sera traité en détail à la section 2.5.1. Toutefois, le langage est intimement lié au modèle d'abstrait. Il est aussi possible que deux langages prennent des approches différentes pour un même modèle. C'est le cas de P4 et de Domino qui sont tous deux basés sur le modèle PISA (voir section 2.5.1). En fait, Domino est basé sur un modèle légèrement différent de PISA (c.-à-d., BANZAÏ). Celui-ci respecte néanmoins le modèle PISA. La distinction entre les modèles sera abordée à la section 2.4.2.

2.4.1 P4 : Programming Protocol-Independent Packet Processors

Le contrôle du plan de données dans un réseau programmable passe par une interface commune entre tous les composants réseau. En 2009, la norme OpenFlow a été proposée pour être cette interface. Pour ce faire, il fallait rajouter à la norme les différents champs d'en-tête pris en charge. Ainsi, si en 2009 il y avait 12 champs pris en charge, depuis 2013 il y en a 41 [1]. En raison de la lenteur de l'amélioration de la norme, Bosshart et coll. ont proposé un langage de programmation de composant réseau indépendant des protocoles [1]. Ainsi, ils ont permis à l'interface d'être indépendante des différents protocoles et de leurs multiples champs d'en-tête. Le langage dénommé *Programming Protocol-independant Packet Processor,* ou simplement *P4,* vise à servir à la fois d'interface de configuration et de langage de programmation pour spécifier le comportement des composants du réseau. Avec P4, on peut définir les protocoles et fonctions pris en charge ainsi qu'ajouter des règles pendant l'utilisation du commutateur. Le schéma ci-dessous montre la distinction entre *OpenFlow* et P4.

P4 se base sur le modèle PISA qui est décrit à la section 2.5.1. Ainsi, il permet de décrire l'analyseur syntaxique (*parser*) ainsi que les différentes tables de comparaison-action (*match-action*). Le modèle de traitement des paquets est montré à la figure 2.3. Dans ce modèle, on distingue bien l'influence de l'architecture PISA. La phase de compilation permet de déterminer un graphe pour l'analyse syntaxique, mais aussi un graphe de dépendance entre les différentes tables. Ainsi, il est possible de déterminer la séquence de traitement des paquets et de paralléliser ce traitement. On notera aussi que la section de classification et recherche

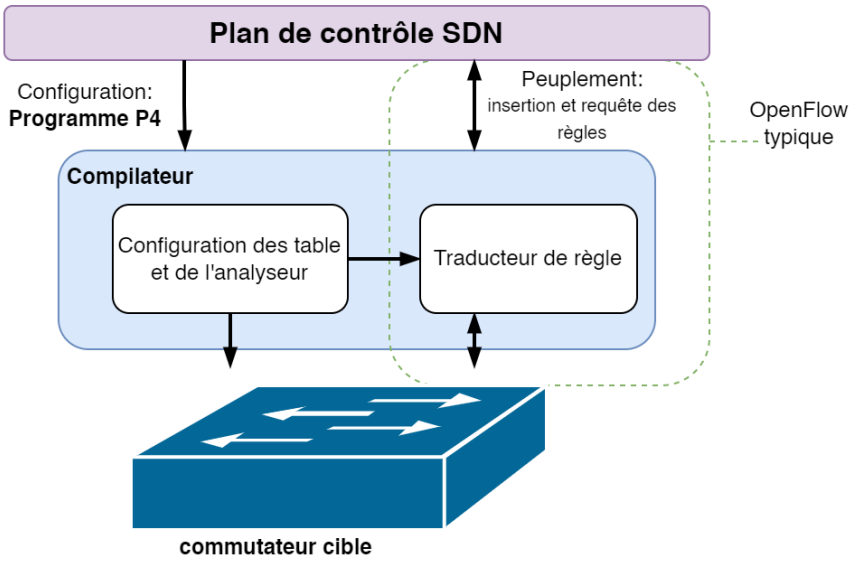


Figure 2.2 Configuration d'un commutateur programmable en P4 (reproduit de Bosshart et coll. [1])

décrite à la section 2.3.2 est jumelée avec la section opération sur les paquets, décrite à la section 2.7.1. Ce jumelage provient de l'architecture PISA. Les architectures ciblées par P4 sont les ASICs, les NPUs, les FPGAs et les commutateurs purement logiciels.

PISCES est une tentative d'implémentation de P4 sur un commutateur logiciel nommé *OpenVSwitch*. L'intérêt d'un commutateur logiciel provient du fait que le nombre d'interfaces virtuelles dans les centres de données dépasse de beaucoup celui des interfaces physiques. En effet, chaque serveur physique peut s'occuper d'un grand nombre de machines virtuelles qui comportent à leur tour un bon nombre d'interfaces réseau virtuelles. Pour gérer le trafic entre les interfaces virtuelles et physiques, il est possible d'utiliser des commutateurs virtuels comme OpenVSwitch.

Cette intégration permet de simplifier grandement l'ajout de nouveaux protocoles, les programmes P4 étant environ 40 fois plus concis que leur équivalent dans le code source original [13]. Or, cette diminution de la taille du code source s'accompagne d'une faible diminution de la bande passante. On parle ici d'une vingtaine de cycles d'opération par paquet, ce qui représente entre 1% et 5% de chaque étape du traitement. Le problème principal dans la création de PISCES était de concilier le modèle de traitement d'OVS avec celui de P4. De plus, pour qu'OVS puisse fonctionner à des débits réseau décents, il dépend grandement

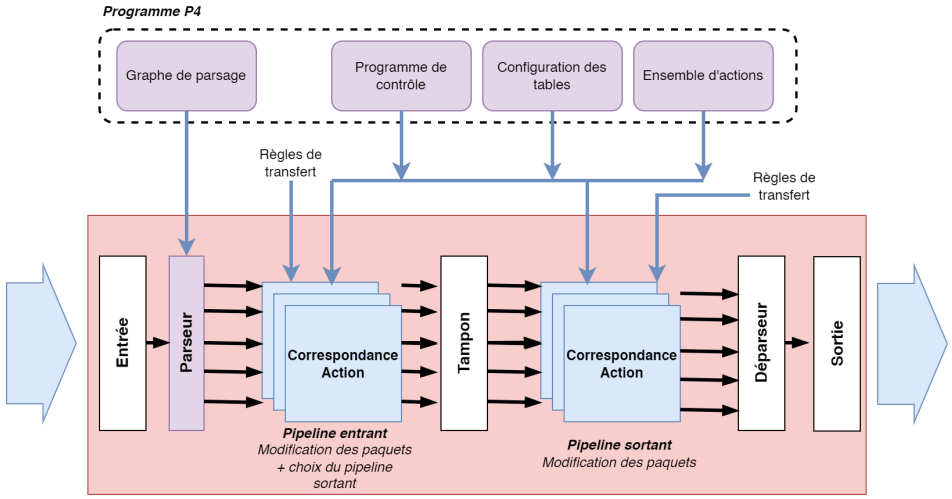


Figure 2.3 Le modèle abstrait du traitement selon le programme P4 (reproduit de Bosshart et coll. [1])

de modules du noyau du système d'exploitation. Cela complique l'adaptation du modèle et empêche également de modifier un programme P4 à l'exécution. En effet, PISCES nécessite d'être recompilé avec tout le code d'OVS pour rajouter un nouveau programme P4. On peut retenir de cette implémentation qu'un programme P4 peut être plus concis et presque aussi performant qu'une implémentation dépendante de protocole d'un commutateur virtuel. Toutefois, il faut que le modèle de traitement du commutateur virtuel corresponde à celui de P4.

2.4.2 Domino

Rendre le réseau programmable signifie aussi de permettre l'ajout d'algorithmes pour le traitement du plan de données. Pour ce faire, P4 se base sur des tables de correspondance-action (*Match-action*). Chaque paquet, une fois analysé, sera classé en fonction des opérations requises qui seront ensuite effectuées. Un algorithme de traitement serait alors une série d'opérations effectuées sur un paquet. C'est ce que les concepteurs de Domino, Sivaraman et coll., appellent une transaction de paquet [4]. Tout le langage Domino est basé sur ces transactions et vise à exprimer les algorithmes de traitement sur le plan des données. Ainsi, il se distingue de P4 puisqu'il ne permet ni la configuration lors de l'exécution ni la modification de l'analyseur syntaxique. La syntaxe de Domino est basée sur le langage C et n'apporte pas beaucoup de nouveauté en elle-même. C'est la compilation, basée sur le modèle BANZAÏ, qui représente le plus grand apport.

Dans ce langage, les algorithmes ou transactions sont décomposés en une séquence d'instructions atomiques qui s'exécutent en un cycle d'horloge. Or, à la différence d'un pipeline d'instruction standard d'un processeur, chaque instruction est effectuée par une unité de calcul différente avec ses propres registres et capacités de calcul. Le jeu d'instruction étant adapté aux algorithmes réseau, il permet de rendre atomiques des opérations qui ne le sont pas sur un processeur RISC standard. Par exemple, il est possible d'executer des opérations jumelées qui permettent de modifier deux champs d'en-tête en même temps après avoir effectué un test logique sur ces deux champs. Bien entendu, les atomes qui prennent en charge les opérations les plus complexes occupent une surface plus grande et donc contraignent la période d'horloge. Par ailleurs, le langage empêche d'utiliser des opérations qui pourraient ralentir le traitement. Ainsi, les opérations comportant des itérations, des flux de contrôle sans structure (goto, break et continue), de la mémoire dynamique et finalement des pointeurs où l'accès à une section non analysée des paquets est impossible [4].

Les auteurs mentionnent dans l'article de référence [4] que la possibilité de prendre en charge plusieurs transactions par commutateur n'avait pas été explorée. Plusieurs transactions impliqueraient de parfois utiliser les mêmes atomes, ce qui pourrait affecter le traitement des paquets. En effet, les états conservés pourraient ne pas s'appliquer aux mêmes transactions. Donc, il faudrait des mécanismes de protection qui pourraient ralentir, voire bloquer, le traitement de certaines transactions [4].

2.5 Architectures configurables

Les langages de programmation présentés à la section précédente ne peuvent modifier les fonctionnalités d'un commutateur que si l'architecture sous-jacente le permet. Cette section présente donc comment les éléments configurables nous permettent d'avoir des commutateurs programmables. Nous commençons par présenter le modèle PISA. Nous présentons ensuite les différents composants programmables d'un pipeline PISA, à savoir, l'analyseur syntaxique, les tables de correspondance-action pour finir avec l'ordonnanceur.

2.5.1 Protocol Independent Switch Architecture

Le modèle PISA est composé d'un analyseur syntaxique programmable et de plusieurs étapes de correspondance et d'action arrangées en série, en parallèle, ou les deux [1]. Ce modèle suit ce qui est représenté à la figure 2.4. La portion tampon est en fait un ensemble de files à

priorité qui permettent d'ordonnancer et de transférer les paquets de la partie entrante vers la partie sortante [3].

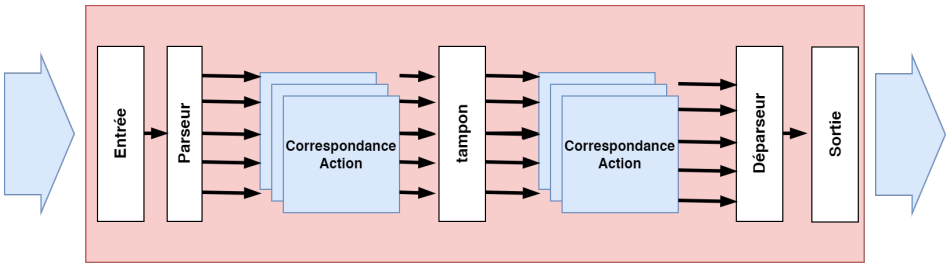


Figure 2.4 Le modèle PISA.

La séparation entre l'analyseur syntaxique et la correspondance (*match*) permet d'avoir un contrôle sur la classification des paquets. En effet, comme il a été mentionné à la section 2.3.2, la classification des paquets se fait souvent en fonction d'un ensemble de règles définies par l'opérateur. À la différence du graphe d'analyse syntaxique, les règles peuvent être modifiées lors du fonctionnement du commutateur. La modification du graphe impose beaucoup de modifications à tous les étages du commutateur. Par ailleurs, les règles de classification sont souvent assez facile à éditer puisqu'il suffit de mettre à jour le contenu de la mémoire RAM et des CAM. La séparation de l'analyseur syntaxique permet également de rendre cette tâche plus simple en réduisant les opérations nécessaires. Effectivement, les opérations complexes comme la recherche et la vérification des sommes de contrôle sont dans la portion correspondance et action. Ainsi, l'analyseur peut se réduire à une machine à état basée sur le graphe de dépendance des protocoles fournis lors de la programmation.

L'architecture considère qu'après l'analyseur syntaxique, les en-têtes des paquets sont acheminés vers les tables sur un vecteur de paquets. Ce vecteur serait en fait un bus au moins aussi large que l'en-tête du paquet. Ainsi, chaque bit peut théoriquement subir des opérations séparées.

BANZAÏ est un modèle basé sur le concept de PISA [11]. Toutefois, ce modèle rajoute un élément au modèle de PISA, l'atome, élément de calcul pour l'exécution d'une instruction faisant partie d'une transaction de paquet. Chaque transaction aurait alors une séquence d'atomes déterminée à traverser. Les atomes s'occupent d'exécuter les actions en fonction des correspondances.

2.5.2 Analyseur syntaxique programmable

Afin d'être indépendant de protocole, il est nécessaire que le modèle PISA comporte un analyseur programmable. L'analyseur identifie les protocoles présents dans le paquet ce qui en expose la structure. L'analyseur doit ensuite rendre disponibles les champs d'en-tête requis pour le traitement. Un analyseur programmable doit pouvoir être configuré pour prendre en charge n'importe quel protocole afin de permettre un commutateur indépendant de protocoles.

La structure de l'encapsulation des paquets impose le parcours des champs d'entête. En effet, les champs d'entête d'un protocole s'analysent couche par couche comme un oignon. La première couche détermine le protocole utilisé pour la seconde couche et ainsi de suite jusqu'à ce que tous les champs utiles soient accumulés. Un analyseur syntaxique programmable est donc une machine à état pouvant parcourir les différentes couches selon les protocoles configurés. La figure 2.5 montre le modèle de ce genre d'analyseur.

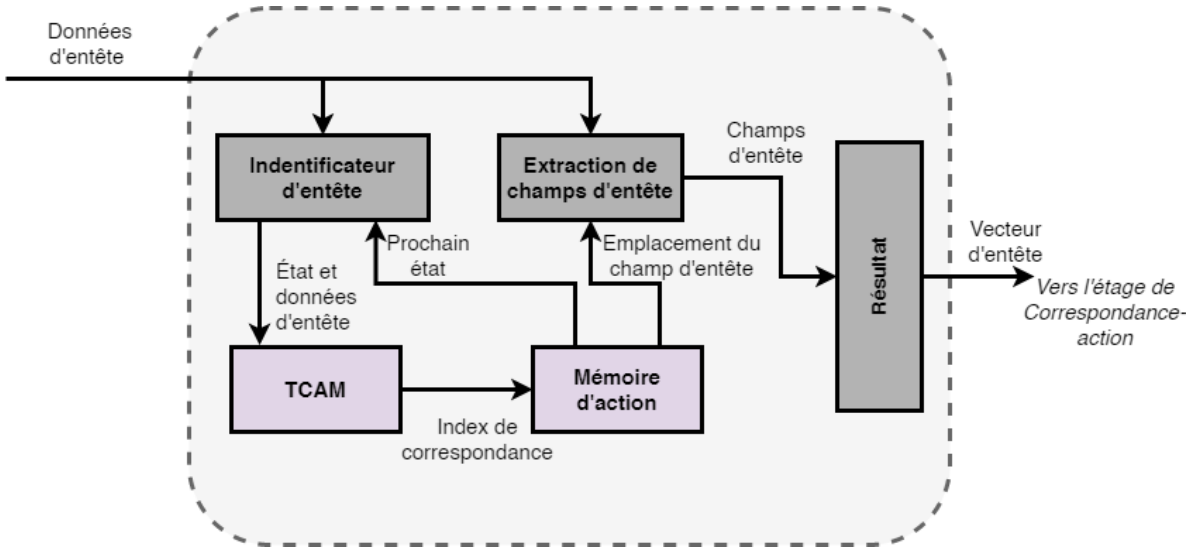


Figure 2.5 Le modèle d'un analyseur syntaxique programmable (Reproduit de Gibbs et coll. [2])

L'identification des protocoles est effectuée au moyen de TCAM qui fait correspondre les codes d'identification de protocoles avec les actions à effectuer. Les actions étant d'accumuler les champs d'entête pertinent pour le traitement qui va suivre et de faire progresser la machine à état. L'état permet ensuite d'identifier les champs d'entête du protocole de la couche suivante et ainsi de suite.

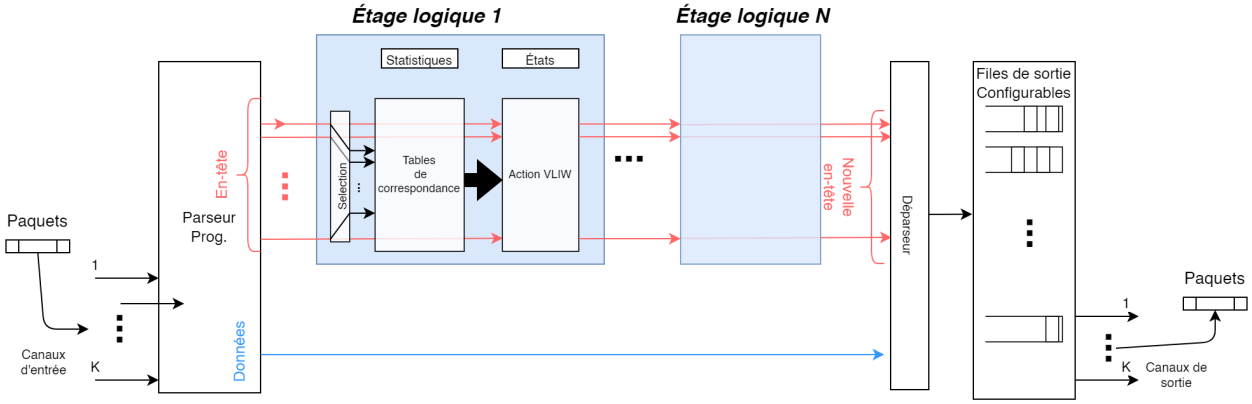
Les états que l'on mentionne ici ont pour objectif de parcourir le DAG de l'analyseur et non de gérer les états d'une connexion encore moins de prendre en charge différents flux. Bien qu'il serait techniquement possible d'effectuer du traitement à états à l'aide de ce composant, ce ne serait ni pratique ni efficace. En effet, plus l'on utilise un nombre élevé d'états plus l'accumulation de champ d'entêtes prendra de cycles d'horloge. Chaque transition d'état prolonge la latence de traitement du paquet en plus de réduire le débit de résultats d'entête. Aussi, il est désirable qu'une séquence de protocoles donnée soit toujours évaluée de la même façon dans l'analyse syntaxique. Cela garantit un débit de traitement pour une pile de protocoles donné indépendamment du flux. En général, il est souhaitable que le traitement à état n'ait pas d'impact sur le traitement sans états.

2.5.3 Tables de correspondance-action reconfigurables

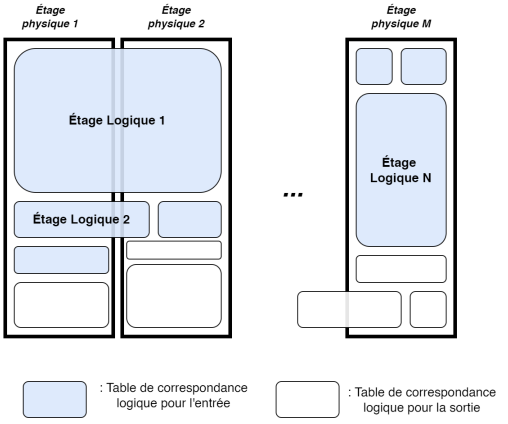
La portion correspondance-action amène le plus de flexibilité dans un commutateur configurable [3]. Cette flexibilité permet de créer de nouveaux algorithmes réseau et ainsi de programmer le comportement des composantes réseau de façon efficace. Un commutateur peut devenir un routeur ou même un pare-feu s'il est bien programmé [3]. Bosshart et coll. ont réussi à créer un commutateur reconfigurable permettant d'atteindre un débit de 64×10Gb/s [3].

Pour permettre de reconfigurer les tables de correspondance-action, les auteurs ont imaginé une architecture complète basée sur de la mémoire RAM, des TCAM et des processeurs VLIW (voir figure 2.6). Chaque étage serait muni d'un processeur, d'un espace RAM et de TCAM, l'espace RAM pouvant servir à plusieurs étages logiques. De plus, une portion des opérations de correspondance peuvent être réalisées au moyen de RAM plutôt que de TCAM en faisant des correspondances binaires basées sur une fonction de hachage. Cette technique permet d'utiliser $6\times$ moins de surface que des TCAM.

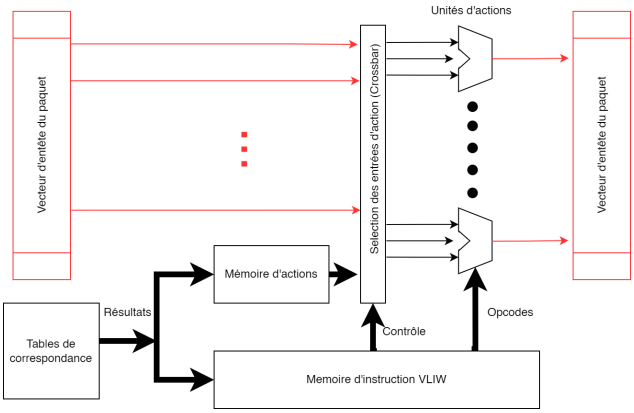
Pour les processeurs VLIW, l'entrée est composée d'un *crossbar* qui permet de décaler des portions d'en-tête. Aussi, le processeur est décomposé en 64 unités de 8 bits, 96 unités de 16 bits et 64 unités de 32 bits. Les petites unités de 8 bits pouvant être combinées pour former des unités de 16 bits. Donc, chaque octet de l'en-tête peut être modifié simultanément. Cela permet de décaler complètement l'en-tête.



(a) Modèle RMT comme une séquence d'étages de correspondance-action



(b) Configuration flexible des tables de correspondance



(c) Architecture des unités d'action VLIW

Figure 2.6 Le modèle d'architecture RMT (reproduit de Bosshart et coll. [3])

L'ajout de machine à états à l'aide des tables de correspondance-action permet aussi d'accroître la flexibilité du commutateur configurable. Les tables contiennent alors les conditions de changement d'états ainsi que leurs transitions correspondantes [15]. Une telle machine à états permettrait d'ajouter des fonctionnalités de gestion de connexions avec états (*Statefull*). La proposition d'utiliser les tables de comparaison-action pour des machines à état provient de la description d'OpenState [15]. Les auteurs d'OpenState ont proposé d'utiliser les tables de comparaison-action pour mémoriser les états et les transitions possibles. Ainsi, ils ont démontré qu'en utilisant l'API d'OpenFlow 1.3, OpenFlow pouvait supporter les machines à états.

2.5.4 Ordonnancement programmable

L'ordonnancement des paquets repose sur deux décisions : *dans quel ordre* et *quand* les transmettre [4]. Pour la plupart des algorithmes d'ordonnancement, il est possible de répondre sans équivoque à ces deux questions au moment où ils sont mis en file. Sivaraman et coll. ont donc créé un système de file qui permet d'implémenter bon nombre d'algorithmes d'ordonnancement de paquets qu'ils soient conservatifs ou non-conservatifs (*work-conserving* ou *non-work-conserving*). Cette structure est basée sur les files *push in, first out* ou PIFO. La PIFO permet d'insérer un élément dans une position arbitraire dans la file en fonction de la priorité de l'élément, mais de toujours enlever les éléments de la file par la tête [4].

Les algorithmes conservatifs gardent le processeur occupé tant qu'il y a quelque chose dans la file. Les algorithmes non conservatifs peuvent forcer le processeur à rester en attente même si la file contient des éléments. Dans le cas d'un algorithme conservatif, on se trouve avec la question « *Dans quel ordre ?* ». Et il est évident que dans le cas d'un algorithme non conservatif, on se trouve avec la question « *Quand ?* ». En effet, si l'on attend un événement particulier pour sortir de file, il est fort probable que le processeur attende.

Pour les algorithmes d'ordonnancement où l'ordre relatif ne change pas avec l'arrivée d'un nouveau paquet, une seule PIFO suffit. Par contre, pour les algorithmes non conservatifs et les algorithmes qui requièrent des modifications à l'ordre relatif, il est nécessaire de composer les PIFO en arbre de recherche. Dans un arbre de PIFO, chaque élément peut être soit une référence vers une PIFO soit un paquet. Dans le cas conservatif, où l'on veut connaître l'ordre d'envoi, l'opération qui détermine l'ordre dans une PIFO est la *scheduling transaction* ou transaction d'ordre. Dans le cas non conservatif, où l'on veut contrôler le moment de l'envoi, l'opération se nomme *shaping transaction* ou transaction de mise en forme. La mise en forme est bien importante puisqu'elle permet de faire respecter des bandes passantes établies en ajustant le débit de sortie. Chaque paquet est ainsi envoyé à un moment précis, calculé pour que le trafic résultant ait un débit donné. L'ordonnanceur est un composant crucial pour la gestion du trafic et la gestion de la qualité des services.

2.6 Gestion d'états

L'identification de flux et la gestion de leurs états permettent un traitement adaptatif de paquet. En effet, le traitement de paquet dépendant de l'état figure dans bon nombre de protocole et d'application (voir section 3.2). La nécessité de mémoriser l'information d'état

pour chaque flux demande un espace mémoire dédié pour chaque flux actif. La cohérence de l'information pose également problème puisque jusqu'à présent les puces de commutateurs programmables ne partagent pas leur mémoire entre les pipelines [16,17]. Pour contourner ce problème de localité les programmeurs de commutateurs utilisent la recirculation de paquet pour rediriger les paquets entre deux pipelines, entrée ou sortie [4,6].

Toutefois certaines modifications à l'architecture PISA pour mieux supporter la gestion d'état ont été proposés. L'architecture Banzai [4] ajoute des opérations de lecture et d'écriture de mémoire à même les composants d'actions du pipeline PISA. Chaque composant d'action et sa mémoire locale composent un atome. Ainsi, ces atomes peuvent effectuer des opérations de gestion d'états. Une modification pour supporter plusieurs pipelines avec l'architecture Banzai a été présentée par Shrivastav [16]. L'architecture de FlowBlaze [5,15,18] apporte un étage dédié à l'exécution d'une machine à états finis étendue (eFSM). Le pipeline est alors une séquence d'étages de traitement avec gestion d'états ou sans gestion d'état. Les étages avec gestions d'état possèdent des tables additionnelles pour la mémorisation des états ainsi que les règles de transition. L'approche de Flowblaze a le mérite de pouvoir réordonnancer les paquets selon les flux. Ce réordonnancement est nécessaire puisque les accès mémoire peuvent prendre plus d'un cycle et ainsi causer des problèmes de cohérence de donnée. En effet, deux paquets d'un même pourraient lire consécutivement la mémoire avant que le premier des deux paquets n'ait pu modifier l'état.

Le coût de la RAM à même la puce du commutateur programmable étant assez élevé, cela contraint la taille de mémoire disponible pour la gestion des états. Plusieurs ont proposé d'utiliser la mémoire RAM de serveur externe pour étendre la mémoire disponible [7,8,19]. Ces approches font usage du protocole RDMA pour utiliser la mémoire de serveurs se trouvant dans le même centre de données. L'approche de Ribosome [8] utilise le commutateur programmable pour faire l'identification des flux et ensuite transfère l'entête vers un processeur de fonction réseau et la charge utile du paquet vers un serveur RDMA. Le processeur de fonction réseau peut être soit un FPGA ou un CPU selon les besoins et effectue seulement le traitement des entêtes ce qui réduit la bande passante nécessaire. Le nouvel entête est ensuite réacheminé vers le commutateur qui va s'occuper de joindre l'entête à la charge utile se trouvant sur le serveur RDMA. Cette approche peut effectuer un traitement à état jusqu'à 1,6 Tb/s [8] en utilisant tofino et des serveurs dédiés.

Les concepteurs de Ribosome, Scazzariello et coll. [8], justifie la nécessité de faire le traitement d'état en dehors du commutateur programmable pour deux raisons. La première, la taille de la mémoire disponible pour un commutateur en ASIC n'est pas suffisante pour la quantité de flux actifs. La seconde, les commutateurs en ASIC ne peuvent pas effectué de mise à jour

fréquente de l'états d'un flux depuis le plan de contrôle. Ils ont utilisé les statistiques de flux obtenues par CAIDA entre 2013 et 2019 [20]. Afin d'établir un lien entre le débit et le nombre de flux actifs, les auteurs ont effectué une régression linéaire sur l'ensemble des traces. Ainsi, ils obtiennent une augmentation de 120 milles flux actifs par Gigabit de trafic.

2.7 Approches d'optimisation

La gestion d'états implique de pouvoir effectuer des actions à partir de l'état et des informations de flux (5-tuple). Parmi ces actions, la mise à jour de l'état est importante, puisqu'elle peut modifier le traitement d'un paquet subséquent dans un intervalle de temps très court. L'emplacement de l'information d'état est donc important puisqu'elle doit être rapidement accessible. On peut distinguer quatre angles d'approche pour permettre la gestion d'états dans un commutateur programmable. Chacune des approches se distingue par le niveau de partage de mémoire nécessaire. Les approches sont couvertes dans un ordre croissant de globalité.

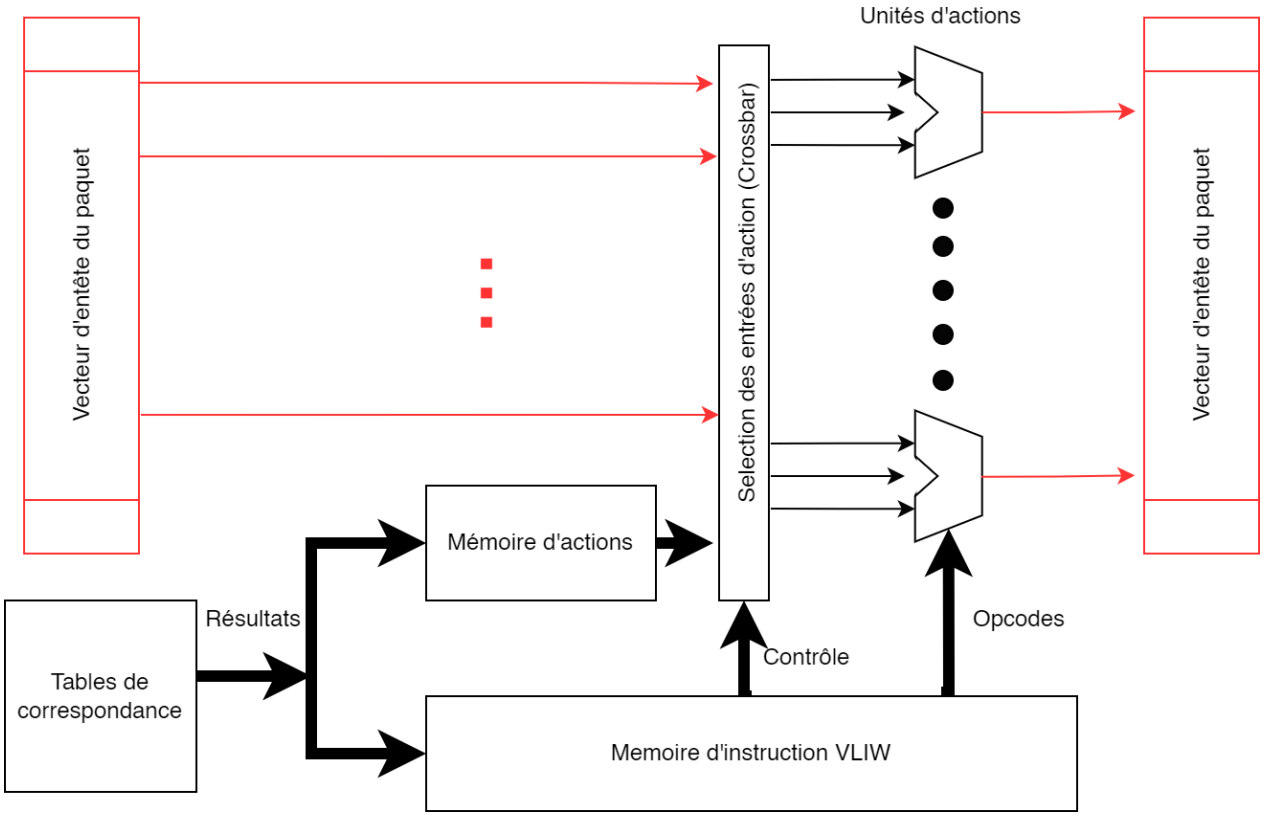


Figure 2.7 Structure d'un processeur VLIW utilisé pour un étage de MAT

La première approche se situe au niveau des unités d'action illustrées à la figure 2.7, qui présente les différentes parties d'un étage de correspondance-action [3]. La portion «tables de correspondance» permet de déterminer le traitement du paquet en comparant aux entrées contenues dans les tables. Ensuite, les paramètres nécessaires au traitement sont transférés à la mémoire d'instruction et à la mémoire d'action. Ces deux mémoires permettent de contrôler les unités arithmétiques qui effectueront des modifications de l'en-tête du paquet. Donc, la seconde approche modifie les tables pour permettre d'y insérer les données d'états. La mémoire d'une table est partagée à l'intérieur d'un étage de pipeline, mais pas entre les étages. La troisième approche se fait au niveau du pipeline, par exemple en modifiant l'ordonnancement des paquets à traiter ou en transférant de l'information d'un pipeline à l'autre. La quatrième approche utilise un composant externe au commutateur pour effectuer la gestion d'état. Il est à noter que ces approches ne sont pas mutuellement exclusives et peuvent coexister dans une architecture. L'approche de Flowblaze [5] en est un exemple, c'est pourquoi cette architecture est discutée dans la section 2.7.1, la section 2.7.2 ainsi que la section 2.7.3.

2.7.1 Composants d'action

Dans un étage de pipeline PISA, tel qu'illustré à la figure 2.7, les opérations sur les entêtes sont effectuées par les unités d'action. Il est donc normal de considérer ces éléments pour exécuter des opérations de modification d'états à partir des données d'entête et autres métadonnées. Sivaraman et coll. [4] ont utilisé cette approche dans leur machine BANZAÏ. La figure 2.8 montre le modèle de la machine BANZAÏ. En effet, chaque composant d'action, appelé atome, se compose d'une petite mémoire et d'une unité fonctionnelle. Cette unité fonctionnelle pouvant effectuer des opérations atomiques de lecture et d'écriture, des opérations à prédicat ou des opérations sans états. Dans leur article, les auteurs présentent différentes variantes d'atomes avec différentes opérations adaptées pour certaines fonctions réseau avec état, par exemple les filtres de Bloom [4]. Toutes ces variantes ont un délai de chemin critique de moins d'une nanoseconde permettant une horloge de 1 GHz. Ce genre de mémoire locale au composant d'action est présent dans le commutateur Tofino ainsi que dans le standard PSA [6] sous la forme de registres disponibles en tant qu'«extern» P4. Toutefois, ces registres offrent moins de possibilités d'opérations que les atomes. Aucune des deux approches n'offre de synchronisation d'état entre différents composants d'action d'un même étage, encore moins une synchronisation d'états entre étages. Afin de transférer l'information d'un étage à l'autre, il faut écrire dans le vecteur d'entêtes puis lire dans un étage subséquent.

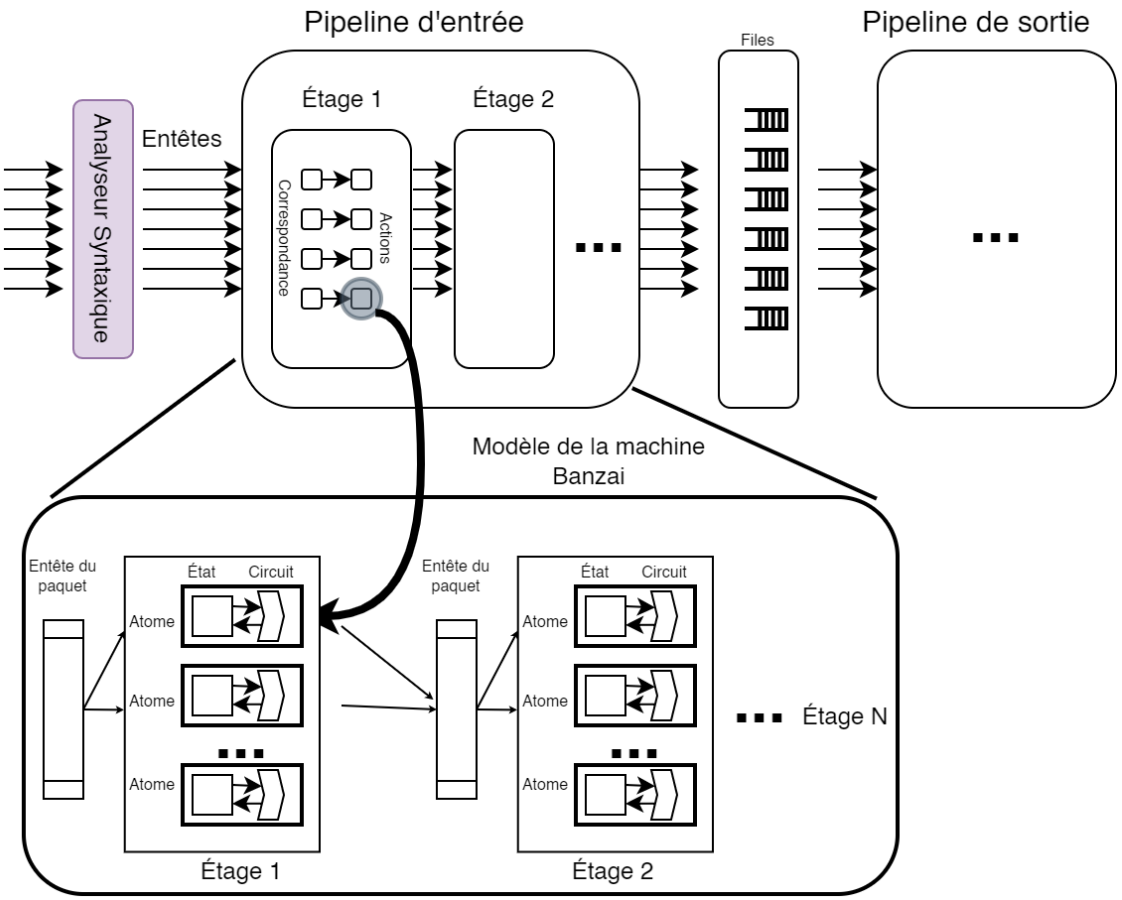


Figure 2.8 Modèle de la machine BANZAÏ (reproduit de Sivaraman et coll. [4])

L'approche proposée par Pontarelli et coll. [5] apporte également des modifications aux composants d'actions pour permettre la gestion d'état. En effet, l'architecture d'un étage de gestion d'états de l'architecture Flowblaze utilise un élément d'action avec un accès bidirectionnel avec la mémoire d'états de l'étage, en A sur la figure 2.9. Toutefois, ils apportent également des modifications au déroulement du pipeline afin de réordonnancer les paquets pour résoudre les dépendances de données causées par ces lectures et écritures. Le bloc B sera discuté à la section 2.7.2, le bloc C quant à lui sera discuté en section 2.7.3.

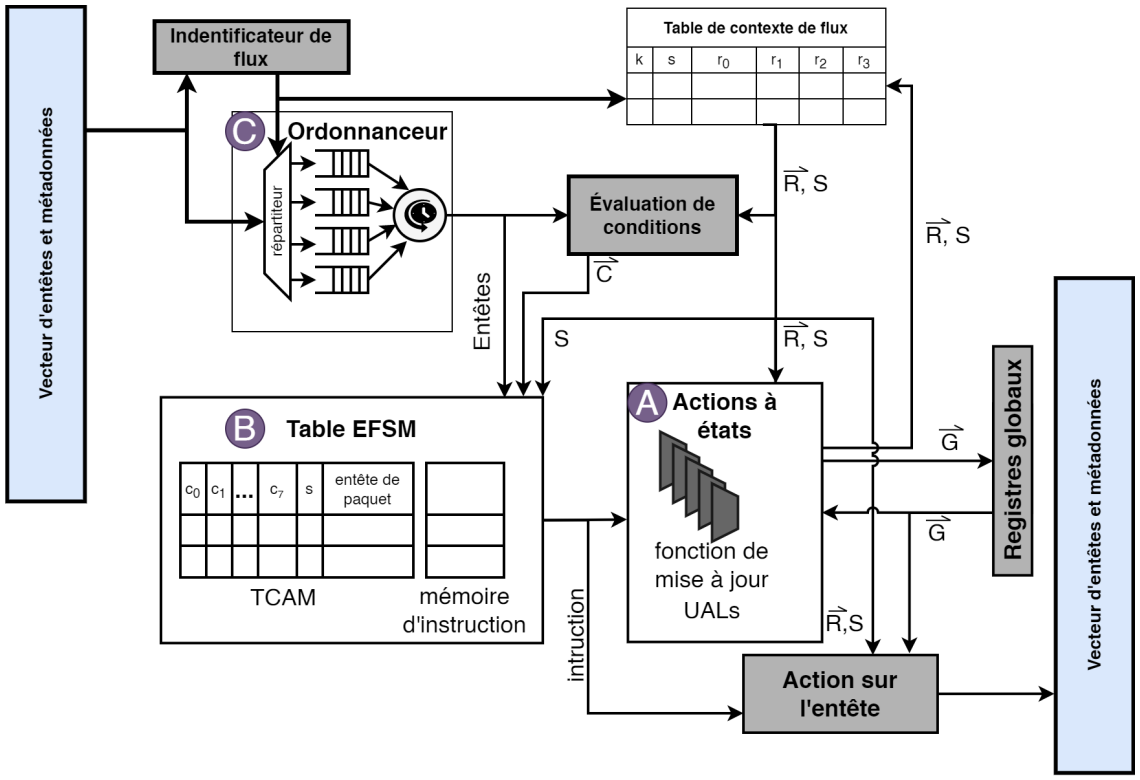


Figure 2.9 Élément de gestion d'états (reproduit de Pontarelli et coll. [5]).
A) Unité d'action avec gestion d'états. B) Table de correspondance pour la gestion d'état d'une machine à état fini étandue (EFSM). C) Ordonnanceur de flux

Les éléments d'actions à états étant au nombre de cinq, c'est également le nombre de mises à jour d'états possibles lors d'une transition d'état. Cette mise à jour peut modifier l'un des huit registres globaux ou bien l'un des quatre registres alloués pour ce flux dans la table de contexte de flux. Chacun de ces registres a une taille de 32 bits. La table de contexte est en fait une table de hachage coucou avec une réserve de huit entrées. L'identification de flux fournit le lien entre les champs d'identification comme le 5-tuple et la clé du flux dans la table de contexte. Notons que la séparation de la table de contexte et la table EFSM séparent les informations d'états des règles de transition. Seule la table de contexte peut être modifiée depuis les éléments d'actions. Les règles de transition d'états sont définies par le programmeur et ne seront pas modifier pour la durée d'utilisation du programme, contrairement au contexte des flux qui seront modifiés à chaque événement de flux.

2.7.2 Tables de correspondances

Une autre approche de traitement d'états peut passer par les tables de correspondances. Ces tables permettent d'identifier l'action à exécuter en fonction d'un ensemble de champs d'en-tête ou de metadonnées. Les tables de correspondances configurables telles que présentées par Bosshart et coll. [3] utilisent différentes structures afin de trouver quelle entrée dans la mémoire de l'étage correspond aux champs. La figure 2.10 montre le plan de contrôle dans l'encadré pointillé. Les étages de correspondance-action reçoivent la configuration depuis le plan de contrôle.

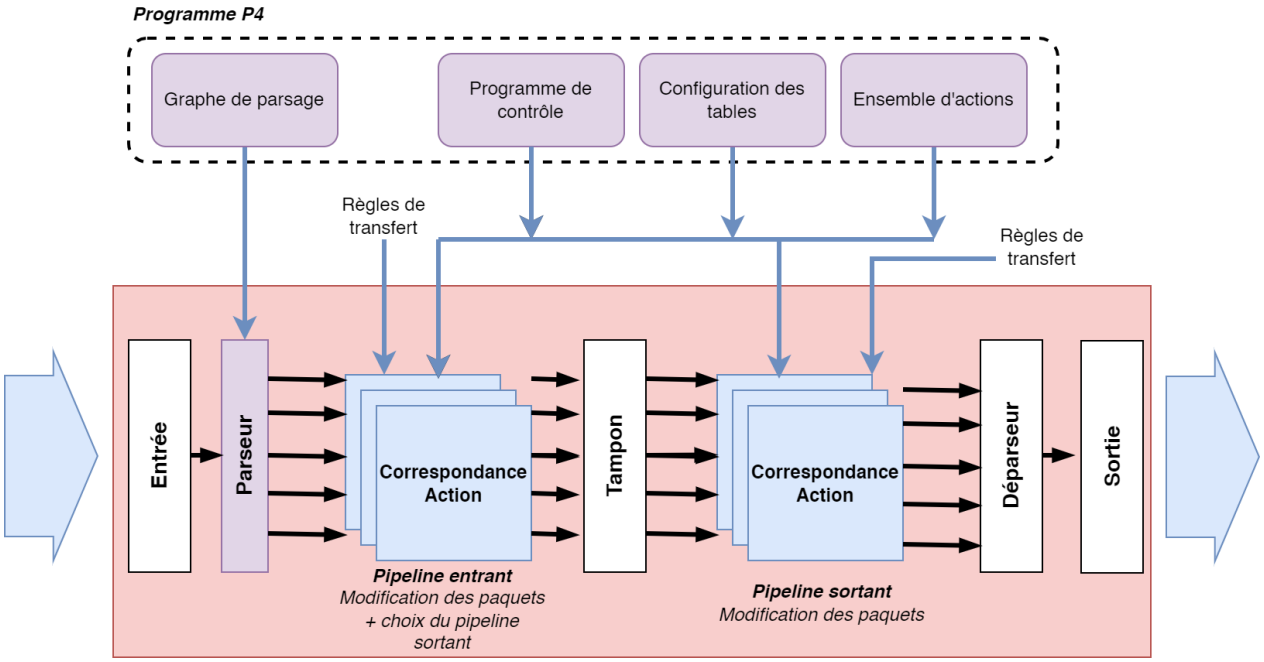


Figure 2.10 Le modèle abstrait du traitement. (reproduit de Bosshart et coll. [3])

Le programme de contrôle effectue les insertions de règles et répond aux requêtes sur le plan de contrôle. Les paquets destinés au plan de contrôle sont minimalement identifiés par le parseur et les tables de correspondances avant d'être acheminés vers le processeur où s'execute le programme de contrôle. Si le paquet reçu par le programme entraîne une modification des règles de transfert, alors le programme fait une mise à jour des tables. Comme cette boucle d'interaction sert au plan de contrôle qui est considéré plus lent, elle n'est pas bien adaptée pour la mise à jour fréquente de règles.

Par contre, faire des mises à jour directement depuis le plan des données peut constituer une

faille de sécurité. Il faut donc faire attention de ne pas pouvoir insérer des règles arbitraires, car de cette manière, un attaquant pourrait dévier le trafic et ainsi effectuer une attaque par personne interposée. L'insertion et l'éviction de règles dans les tables doit rester sous le contrôle du programmeur et ne pas empiété sur les autres fonctions du commutateur.

Intéressons-nous maintenant à l'architecture de ces tables de correspondance. La figure 2.11 montre l'organisation de la mémoire RAM du commutateur en fonction des différents étages. Chaque table définie dans le programme du commutateur aura un étage logique correspondant et une taille déterminée à la compilation [3].

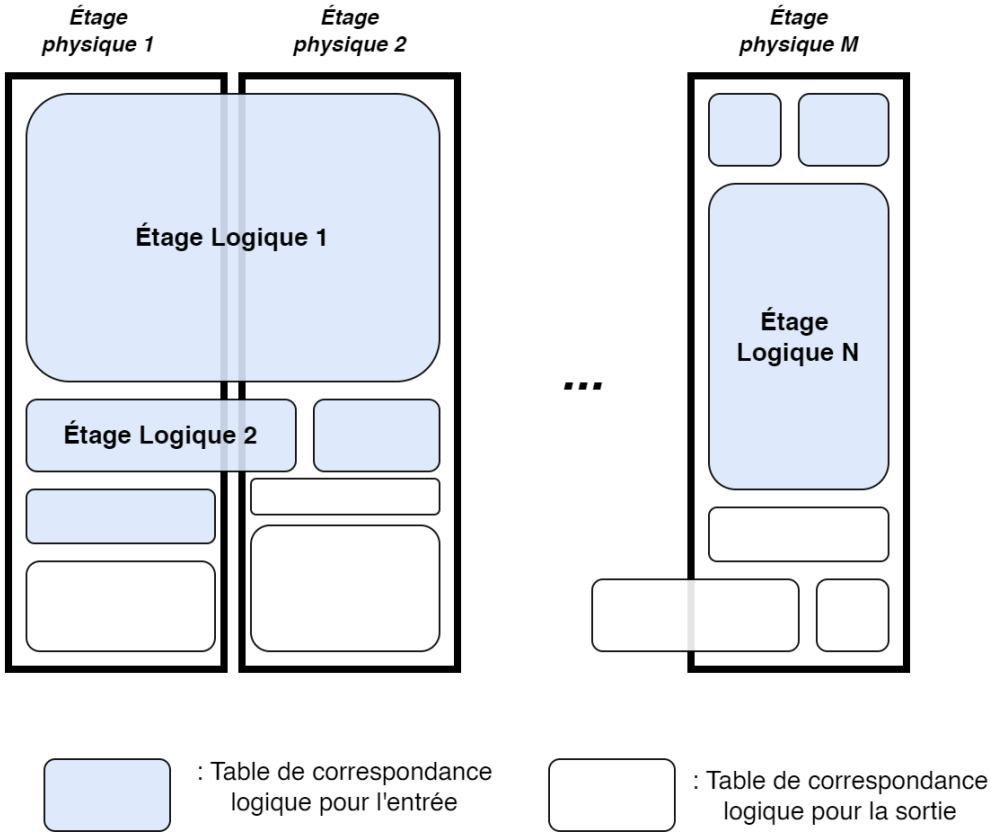


Figure 2.11 Configuration flexible des tables de correspondance (reproduit de Bosshart et coll. [3])

Comme les tables permettent d'identifier l'opération à effectuer dans un étage donné, une table peut comporter les différentes actions associées aux différents états d'un protocole. Toutefois, pour traiter plusieurs flux avec des états indépendants, il faut retenir l'état courant en relation avec les champs l'identifiant comme le 5-tuple. Une fois le paquet courant identifié comme appartenant à un flux préétabli, alors on peut chercher l'état du flux. Dans

les applications comportant une grande quantité de flux, l'utilisation de tables de hachage pour faire une correspondance exacte est généralement préférable à l'utilisation de TCAM, en raison de la complexité du circuit. Pour des applications où l'état n'est que fonction de la présence d'une connexion établie, comme un pare-feu ou un répartiteur de charge, il n'y a généralement pas besoin de variable d'états puisque la présence de la règle confirme l'état de la connexion. De plus, dans le cas des pare-feux, les règles doivent être insérées avant qu'une réponse arrive. Toutefois, cette réponse va prendre le temps d'un aller-retour avec le serveur avant de revenir vers le pare-feu. Ainsi, même avec le lent plan de contrôle, le pare-feu peut faire des mises à jour.

Dans le cas de Flowblaze [5], qui se trouve à la figure 2.9, on retrouve une modification d'un étage de correspondance actions qui permet une écriture dans les tables depuis le plan de donnée. Toutefois, seule la table de contexte peut être écrite de la sorte. La table EFSM ne sert qu'à contenir les règles de transition d'états et n'a donc besoin que d'être configurée une seule fois.

En insérant un composant entre le programme de contrôle et le processeur réseau, il est possible de faire des insertions pour faire en sorte que la mise à jour des états puisse prendre le moins de temps possible et ainsi garantir un traitement approprié. Cela soulève la question : « *Quelle est la fréquence de mise à jour des tables ?* ». Cette question sera étudiée au chapitre 4 pour le cas de TCP. La fréquence de mise à jour dépend de l'application. Scazzariello et coll. estiment le nombre de nouveau flux actif par seconde à 4 000 flux pour chaque Gb/s de débit. Ils estiment également le nombre de flux actifs à 120 milles flux pour chaque Gb/s de débit. Ainsi, pour un processeur de commutateur à 6,5 Tb/s comme le Tofino, cela signifierait 26 millions de nouveaux flux par seconde [8] et 780 millions de flux actifs. Chaque flux actif nécessite un espace mémoire pour l'identification des flux, donc dans le cas d'un 5-tuple IPv4 on a 13 octets d'identification ce qui pose la mémoire nécessaire uniquement pour l'identification à 10 GB. Toutefois, ce même processeur ne pourrait supporter qu'environ cent mille modifications par seconde et n'a qu'environ 10-100 MB de mémoire SRAM [8, 21, 22]. Le nombre de nouveaux flux par seconde peut nous donner une idée du nombre d'entrées dans les tables, mais ne nous informe pas de la fréquence de mise à jour de ces entrées. Aussi, ce nombre ne fait pas de différence entre les différents protocoles utilisés.

2.7.3 Approches au niveau du pipeline

Le modèle PISA décrit la composition d'un pipeline, mais les architectures vont généralement utiliser plus d'un pipeline. Cette approche multipipeline attribue un certain nombre de ports

pour chaque pipeline d'entrée et de sortie. Une fois passé par le pipeline d'entrée, il faut réacheminer le paquet vers le pipeline de sortie qui correspond au port de sortie en passant par l'ordonnanceur programmable. Dans le cas d'un paquet destiné au processeur, il faut quand même passer par l'entrée et la sortie afin de transférer le paquet vers le port du CPU. Tous les trajets de paquets permis selon le standard PSA [6] se trouvent à la figure 2.12. Ce sont également les trajets possible dans le processeur Tofino [22].

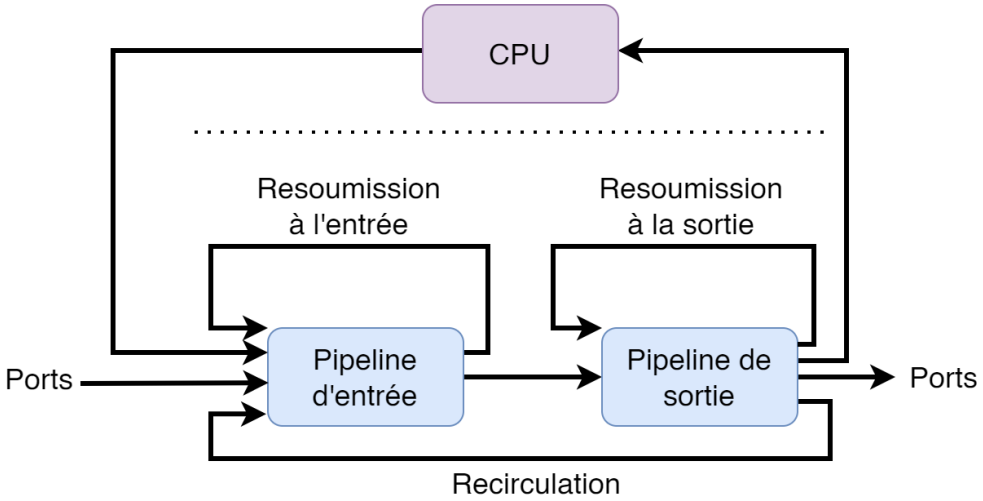


Figure 2.12 Trajets des paquets entre les pipelines selon le standard PSA [6]

La resoumissions et la recirculation sont très pratiques pour le traitement à états, en effet elles permettent de transférer des informations sous forme de métadonnée dans le vecteur d'entête d'un pipeline à l'autre ou entre les étages. Cela a toutefois pour effet d'augmenter la latence de traitement et de diminuer le débit de traitement. En effet, le paquet recirculé devra parcourir plusieurs fois les pipelines. À chaque nouveau pipeline parcouru, le délai de traitement augmente et l'on réduit le nombre de paquets traité. Si l'on recircule trop souvent les paquets on court le danger de causé de la congestion [6].

La resoumissions est particulièrement pratique dans les applications de tunnel [6, 22]. Un tunnel encapsule les paquets à l'entrée du tunnel en ajoutant des protocoles nécessaires pour acheminer le paquet jusqu'à la sortie. Prenons par exemple le cas de GTP, un paquet TCP/IP arrivant à l'entrée du réseau cellulaire sera placé par-dessus l'entête GTP-u. L'entête GTP-u sera toutefois acheminé dans le réseau cellulaire au moyen de TCP/IP. Ainsi, il y a deux entêtes TCP/IP dans les paquets, une version qui est dans la charge utile du paquet et une version qui encapsule cette charge utile et ne sert qu'à l'intérieur du réseau cellulaire. Dans le cas de la sortie du tunnel, l'entête externe pourra être traité par un pipeline supportant

TCP/IP qui identifierait que le paquet est arrivé à la fin du tunnel. Une fois cette identification faite le paquet situé dans la charge utile pourra être resoumis et seulement à ce moment on pourra identifier le port de sortie du tunnel en fonction de l'adresse IP de destination valide à l'extérieur du réseau cellulaire. Cela permet de réutiliser une bonne portion du programme de traitement de TCP/IP pour l'entête extérieur et l'entête intérieur. Ce genre de tunnel porte également le nom de réseau virtuel puisqu'il permet de faire abstraction du réseau. Une autre application de la resoumissions est de faire de faire de la spéculation de traitement et de resoumettre en cas d'erreur [22]. À la différence de la recirculation, la resoumissions n'utilise pas l'ordonnanceur de trafic. Ainsi la recirculation peut mettre à profit l'ordonnanceur pour résoudre des problèmes de dépendance entre paquets d'un même flux. C'est d'ailleurs pour cette raison que les concepteurs de Flowblaze [5,23] on intégré un ordonnanceur à même leur élément de gestion d'état. Cela permet également d'augmenter le délai entre le traitement de deux paquets d'un même flux. Ainsi, on peut accommoder de longs accès mémoire.

2.7.4 Approches externes

Puisque la mémoire est contraignante pour les applications à états et puisque la mémoire sur puce est coûteuse, il serait probablement plus économique d'utiliser de la mémoire disponible sur des serveurs via RDMA. En effet, la mémoire sur puce à titre d'exemple, sur une puce de 826 mm^2 utilisant une technologie à 7nm on ne pourrait pas avoir plus de 5GB de mémoire SRAM et ce en ignorant tout port d'entrée sortie et tampon [8]. Il s'agit d'une petite quantité de mémoire comparativement à la mémoire RAM disponible dans les serveurs qui peut monter jusque dans les téraoctets. Cette approche utilise un commutateur programmable pour effectuer les opérations sans états et sert de point d'entrée pour le trafic. À ce commutateur s'ajoute un nombre de serveurs connectés aux ports du commutateur qui rendent disponible une portion de leur mémoire [7,8,21]. Kim et coll. [7] ont proposé cette approche afin d'étendre la capacité des tables présente sur le commutateur. Leur approche implémente un système de cache avec réserve qui effectue une requête RDMA en cas de défaut. Cette approche est illustrée à la figure 2.13.

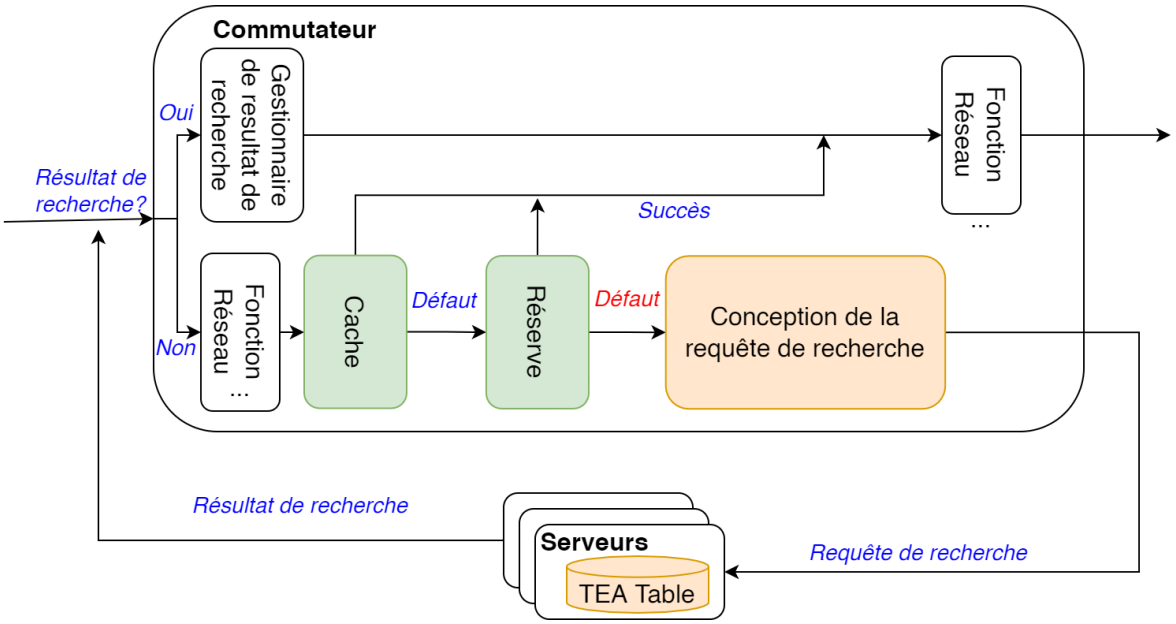


Figure 2.13 Modèle de traitement de TEA [7]

Toutefois, Scazzariello et coll. soutiennent que l'approche de TEA n'est pas bien adaptée pour le traitement à état [8]. Ils avancent que TEA ne peut pas supporter de l'ordonnancement complexe au niveau des flux. Ils démontrent également que l'approche de TEA est assez limitée pour faire face à une haute fréquence de modification de l'état des flux. En effet, les insertions de résultat dans la réserve et la cache passent par le plan de contrôle ce qui ralentit énormément les insertions. L'approche qu'ils proposent délègue complètement le traitement des entêtes vers un serveur dédié. Cela permet de faire le traitement à état au moyen d'un système dédié aux fonctions réseau sur CPU, ASIC ou FPGA. De plus, le commutateur sépare l'entête de la charge utile ce qui a pour effet de maximisé la bande passante à l'entrée des fonctions réseau.

La figure 2.14 montre le modèle de traitement de Ribosome tel que conçu par Scazzariello et coll. [8]. Le surcoût causé par l'utilisation de RDMA pour la charge utile est partiellement compensé par l'augmentation du nombre de paquets traité par seconde via le serveur de fonction réseau. En effet, comme l'entièreté du débit entrant dans la fonction réseau est utilisée pour les entêtes cela permet d'augmenter le nombre de paquets traité par seconde.

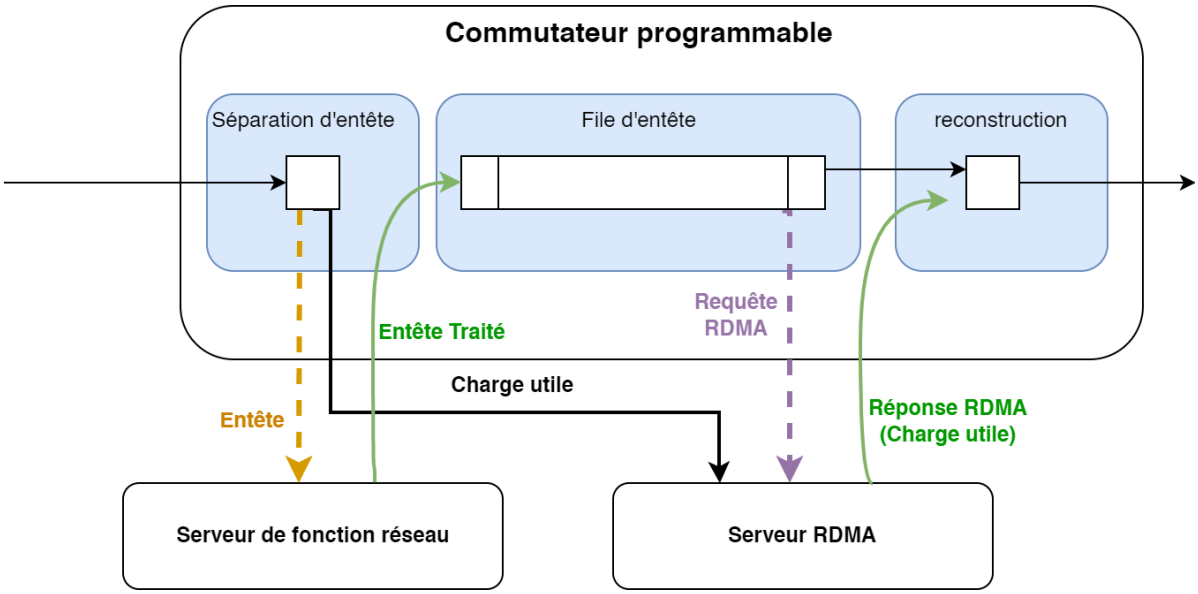


Figure 2.14 Modèle de traitement Ribosome [8]

Il faut noter que l'approche externe ne tente pas de résoudre le problème de la gestion d'état en matériel, mais bien de le contourner, contrairement aux approches présentées aux sections 2.7.1, 2.7.2 et 2.7.3. Ces approches pourrait d'ailleurs être combiné en une seule hiérarchie de mémoire afin de permettre un contrôle sur la localité des informations.

2.8 Conclusion

Les infrastructures réseau étant mises à rude épreuve par l'infonuagique, les objets connectés et les appareils mobiles, nécessitent une évolution dans la manière les concevoir. Les réseaux définis par logiciel, basés sur la séparation des plans réseau, permettent de mieux s'adapter aux différentes situations en permettant un contrôle logiciel du plan de contrôle. Toutefois, l'approche logicielle dépasse maintenant la simple configuration avec l'arrivée de langages de domaine spécifique tel que P4 et Domino. Ces langages vont de pair avec l'arrivée de matériel réseau programmable. Grâce aux tables de correspondance-action, à l'ordonnancement programmable et aux parseurs configurables, l'architecture PISA peut traiter n'importe quel protocole, si le programme décrit sa structure et son traitement. Ainsi, l'architecture PISA est indépendante des protocoles. Maintenant, il est possible d'utiliser un langage de domaine spécifique pour décrire le traitement d'un protocole, le déployer à l'échelle d'un centre de données et d'adapter ce déploiement en fonction de l'état du réseau.

CHAPITRE 3 APPLICATIONS AVEC GESTION D'ÉTATS

Tel que présenté à la section 2.3.1, il existe deux classes de protocoles : les protocoles avec et sans états. Un protocole ou une application avec états doit son nom à sa capacité de rétention de l'information nécessaire au traitement. Ce chapitre porte sur une analyse de l'impact matériel du traitement à états dans différents cas d'application. La première section définit certains concepts nécessaires à l'analyse des cas d'application. La seconde section couvre six cas d'application analysés sous l'angle de la gestion d'états. La troisième section couvre les approches d'améliorations possibles pour l'optimisation d'un processeur réseau programmable de type PISA pour la gestion des états.

3.1 Gestion d'états

Les protocoles avec états impliquent une modification du traitement des paquets dans le temps. Ils impliquent aussi une progression logique entre les états. Cela veut aussi dire que le traitement dépend du paquet entrant et de l'état mémorisé. Ainsi, les protocoles à états nécessitent plus de mémoire que les protocoles sans états. Étant donné la mémoire limitée d'un commutateur réseau, le nombre de flux qu'il peut traiter est lui aussi limité.

3.1.1 Niveaux d'abstraction d'états dans le réseau

Plusieurs niveaux d'états existent dans les réseaux en fonction du point d'observation. En effet, on peut se concentrer sur les flux de paquets ; ainsi, on parle de l'état du flux. Les états se définissent alors en fonction du protocole du flux. On peut aussi se concentrer sur le composant réseau qui effectue le traitement. Dans ce cas, on parle d'état global du composant, puisque l'état a un impact sur l'ensemble des paquets traités, indépendamment du flux. On peut aussi se concentrer sur un ensemble de composants réseau qui sont liés de façon logique. Avec l'arrivée des réseaux définis par logiciel (SDN), l'idée de considérer plusieurs composants réseau comme un ensemble gagne en popularité. Traiter une portion du réseau comme étant un seul macro-composant réseau, *one big switch*, nécessite une gestion d'états distribués parmi plusieurs composants [19]. Considérer un ensemble de composants réseau permet de mieux répartir la charge du trafic. Dans le cas de protocoles avec états, cela veut dire de pouvoir répartir les flux en fonctions de la mémoire et du débit disponibles.

Un exemple de l'abstraction d'états dans le réseau consiste en l'utilisation de réseaux virtuels

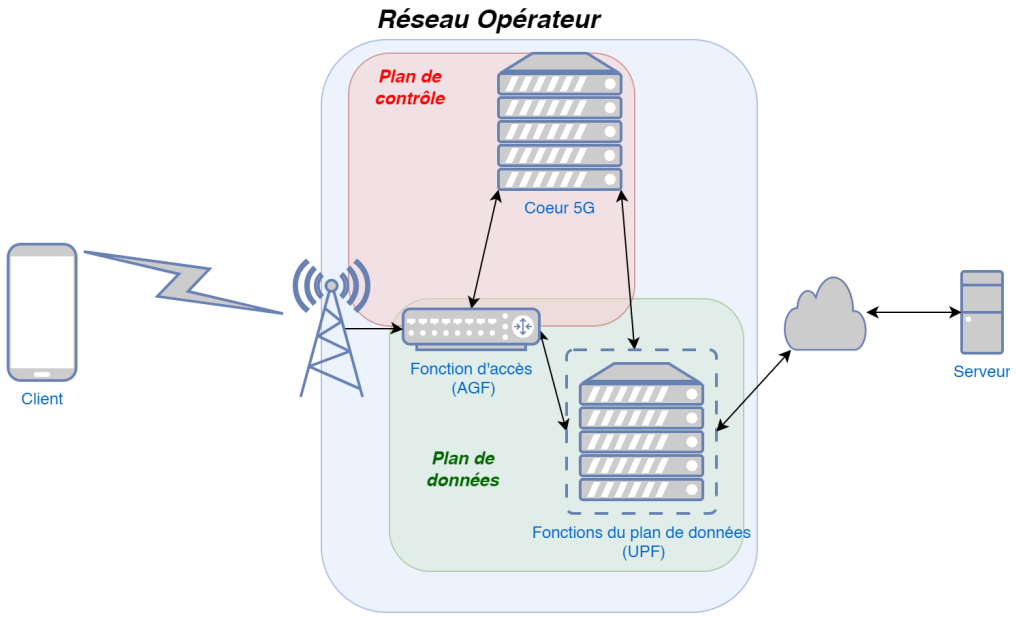


Figure 3.1 Structure du réseau 5G

dans le cadre des réseaux cellulaires 5G (illustré à la figure 3.1). Prenons l'exemple d'un paquet d'une connexion TCP/IP se dirigeant vers un serveur depuis un cellulaire. L'appareil doit d'abord demander accès au réseau en passant par une fonction d'accès qui conserve une table des clients ayant demandé et obtenu accès. Toutefois, lorsqu'un client ne se trouvant pas dans la table demande accès, une vérification sera faite auprès du cœur du réseau. Le cœur du réseau cellulaire conserve l'état des comptes des utilisateurs et peut donc valider la demande d'accès. Une fois l'accès établi, la fonction d'accès s'assure de transférer toute connexion entrante du client vers le plan des données. La fonction d'accès est le point de départ du tunnel GTP, dont il sera question à la section 3.2.4. Ce tunnel vient encapsuler la connexion TCP/IP pour le transfert dans le réseau de l'opérateur jusqu'au point de sortie du tunnel. Le protocole GTP a ses propres états de communication qui contrôlent le transfert dans le domaine de l'opérateur, tandis que le protocole TCP contrôle le transfert entre les deux extrémités du réseau, soit la source et la destination.

Il est aussi possible de distinguer différents niveaux d'états à même un protocole. Un protocole avec états peut avoir une machine à états avec des états bien précis, comme TCP. Toutefois, d'autres informations affectent le traitement et elles évoluent dans le temps. Citons par exemple les informations nécessaires pour la gestion de la congestion pour modifier le taux de transfert de données. Le cas de TCP sera couvert en détail en section 3.2.1 et au chapitre

4, mais l'on peut considérer deux autres valeurs comme étant des états. Ce sont les valeurs de synchronisation et les valeurs de gestion de la fenêtre.

Ainsi, la notion d'état dans le réseau dépend du point d'étude, que l'on se trouve au niveau de l'opérateur ou que l'on regarde à même un protocole. Les défis sont similaires entre les niveaux d'abstraction, mais les solutions adaptées varient. Par exemple, les problèmes de synchronisation d'états sont différents si l'on regarde la cohérence de la mémoire au niveau d'un processeur réseau versus la cohérence de la mémoire d'une base de données répartie dans le coeur d'un réseau cellulaire.

3.1.2 Session et flux

Il est nécessaire de faire une distinction entre une session et un flux. Le terme flux possède une définition assez vague dans la littérature. Parfois, il signifie une connexion où s'échangent des paquets et parfois, il représente un ensemble de paquets ayant des caractéristiques communes comme lorsque l'on parle de 5-tuple. On parlera ici d'une session pour signifier un ensemble de paquets appartenant à une connexion établie d'une application donnée. En effet, dans plusieurs applications, une session pourrait avoir un 5-tuple qui se modifie au fil de la communication en ayant la même connexion d'un point de vue des états du système, par exemple dans les cas d'itinérance réseau. L'établissement d'une session implique au minimum deux états, à savoir lorsque la connexion est établie et lorsqu'elle ne l'est pas. Dans le cas de TCP, il faut que la source et la destination passent par deux états au minimum avant d'atteindre cet état. Ces états sont différents selon que l'on regarde la source ou la destination. Toutefois, TCP est bidirectionnel et peut aussi se trouver dans un état d'ouverture partiel. Dans cette situation, une des deux directions a signalé ne plus vouloir envoyer d'information et donc ne fera qu'envoyer des messages d'accusé de réception jusqu'à la terminaison passive ou active de l'autre extrémité. Comme les deux directions sont quasi-indépendantes, elles sont souvent considérées comme deux flux opposés. Chaque direction comporte un 5-tuple avec les sources et destinations inversées. On peut vouloir traiter une direction différemment d'une autre, par exemple dans le cas des pare-feux à états. Dans ce type d'applications, la direction provenant de l'intérieur du réseau délimité par le pare-feu sera considérée sécuritaire, contrairement à une connexion depuis l'extérieur. Ainsi, une communication sera établie si l'initiation de la connexion provient de l'intérieur, permettant au pare-feu d'autoriser la réponse de la destination. Les deux directions ne respectent pas les mêmes règles de pare-feu.

3.2 Cas d'application

Maintenant que le concept d'état a été présenté, nous pouvons nous intéresser aux applications qui nécessitent ce type de traitement. Les six cas d'applications qui suivent couvrent plusieurs contraintes variées et plusieurs degrés d'abstraction différents afin de couvrir le plus de cas possible. Cela nous permet de comprendre les caractéristiques communes des applications à gestion d'état.

3.2.1 TCP

Le protocole TCP [9] possède trois phases. Chacune de ces phases correspondent à un état spécifique et permet un certain de nombre de transitions, tel qu'illustré à la figure 3.2.

TCP est la base de bons nombres de protocoles d'application, notamment HTTP, RTP, DNS et SMTP. Ces trois phases permettent d'établir la connexion entre le client et le serveur pour s'assurer que la communication soit fiable. Une fois la connexion établie, les sommes de contrôle et les numéros de synchronisation partagés à chaque message permettent au client et au serveur de s'assurer de la séquence des paquets reçus et de détecter les paquets manquants ou endommagés. Il existe plusieurs variantes de TCP, telle que TCP new Reno, qui modifie le comportement du protocole sans modifier les états de bases.

Dans l'en-tête TCP, présenté en figure 3.3, seuls les champs de ports sont nécessaires à l'identification du flux. Rappelons qu'un 5-tuple est composé de ces deux ports, des adresses IP source et destinations et du numéro de protocole contenu dans l'en-tête IP. Ces champs servent à identifier le flux et donc il faut soit les retenir tels quels ou utiliser un hachage de ces champs. Les champs comportent un total de treize octets pour des adresses IPv4 et 37 octets pour des adresses IPv6. L'utilisation d'une table de hachage permet de compresser les champs du 5-tuple en une valeur d'empreinte, la taille de l'empreinte pouvant varier selon la méthode utilisée ou l'application visée. Toutefois, il est nécessaire d'effectuer le hachage du 5-tuple lors du traitement du paquet, ce qui utilise des étages du processeur réseau. Dans le cas des processeurs PISA, ce sont les tables de correspondance qui servent à contenir les empreintes des connexions courantes. La simple identification d'un flux nécessite déjà une portion mémoire, mais le traitement est indépendant du flux. Effectivement, l'opération de hachage sera la même, peu importe le paquet en entrée, mais l'empreinte sera différente.

Le numéro de séquence et le numéro d'accusé de réception servent à valider la réception de données sur une direction. Chaque direction possède son propre numéro de séquence et

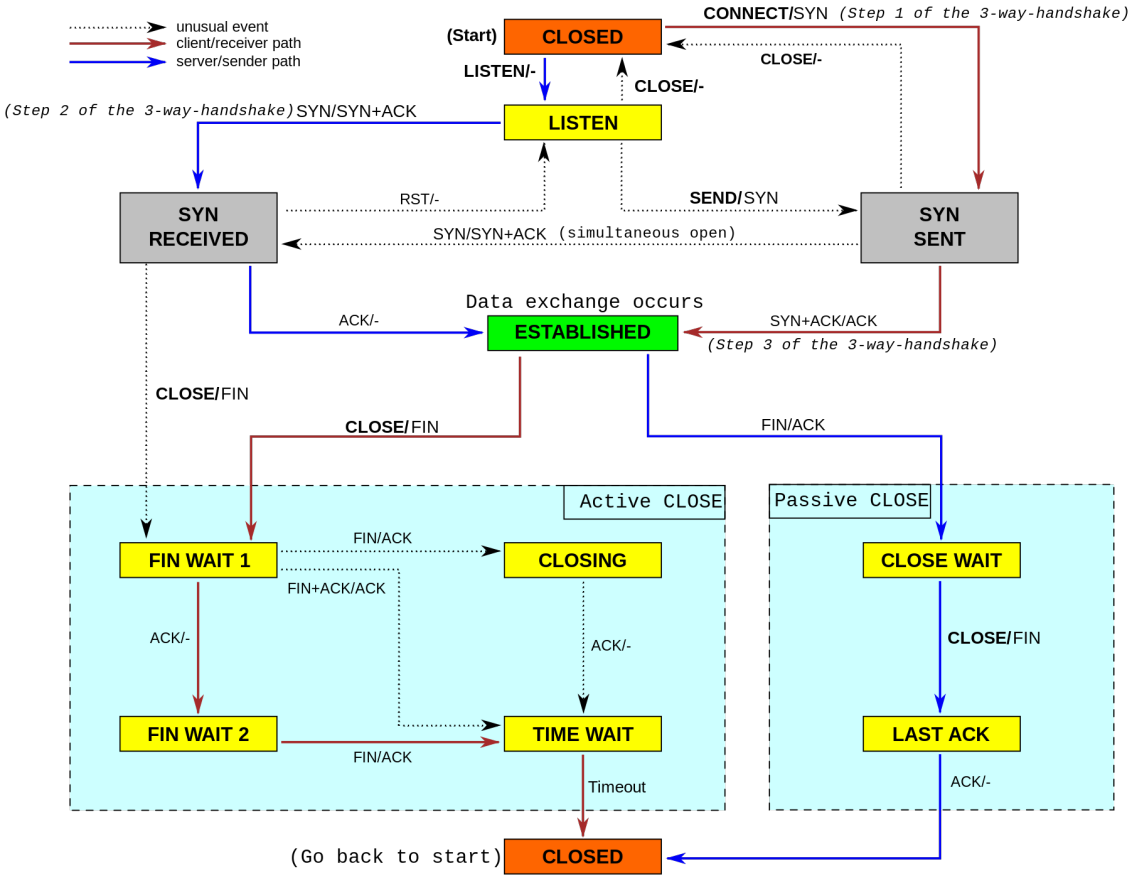


Figure 3.2 Diagramme d'état de TCP [1]

l'incrémente au fur et à mesure que son correspondant lui envoie des accusés de réception. La réception ou non de l'accusé de réception détermine si l'on doit retransmettre un paquet ou non. On peut alors voir le numéro de séquence comme l'état de la séquence de transfert. En plus du numéro de séquence, la taille de la fenêtre de réception est aussi une valeur qui évolue afin de contrôler le taux de transfert. Cette taille est dépendante d'une valeur d'échelle établie avec le premier paquet sur un octet, ainsi que le champ d'en-tête comportant la taille de la fenêtre sur deux octets. La gestion du numéro de séquence requiert quatre octets en mémoire pour chaque direction. Pour ce qui est de la gestion de la fenêtre de réception, la taille de la fenêtre nécessite de décaler vers la gauche la valeur de la taille par la valeur d'échelle. Toutefois, comme la valeur d'échelle reste constante tout au long de la connexion, seuls quatre octets sont nécessaires pour la modification.

1. par Sergiodc2, Marty Pauley, Scil100 CC BY-SA 3.0.
https://commons.wikimedia.org/wiki/File:Tcp_state_diagram_fixed_new.svg

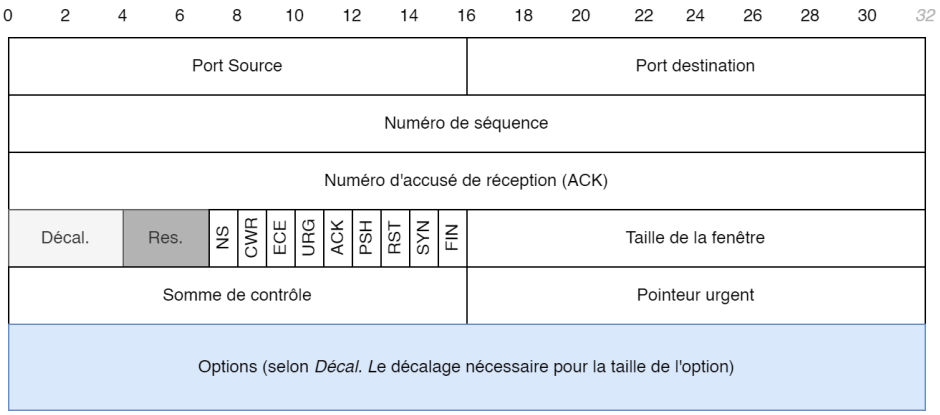


Figure 3.3 Structure de l'en-tête TCP (inspiré de [9])

S'il est possible d'encoder l'état de la connexion TCP sur un seul octet (représentant les douze états de la figure 3.2), la gestion d'autres aspects de la communication nécessite plus de mémoire.

3.2.2 Tempête de SYN

Un autre cas d'application provient de la détection d'attaques de déni de service. Les attaques par déni de service et parfois même des défaillances réseau peuvent être causées par des requêtes frauduleuses, par exemple, les requêtes SYN du protocole TCP. Dans ce genre d'attaque, une grande quantité de paquets TCP vient faire déborder les capacités de traitement des composants réseau. Ces paquets sont basés sur un état spécifique de la connexion TCP, la demande de synchronisation ou SYN. Toutefois, tous les états de création de la connexion sont omis. Ainsi, sans gestion d'état, un commutateur tenterait de traiter tous les paquets et succomberait à l'attaque. À l'inverse, s'il est capable de détecter que le paquet n'appartient pas à un flux préétabli ; les paquets seraient sommairement traités puis rejetés, réduisant grandement les ressources utilisées. Aussi, comme il a été mentionné dans la section 3.2.1, l'utilisation d'une fonction de hachage permet de compresser la taille du 5-tuple en une empreinte et donc de réduire l'espace mémoire nécessaire. Dans le cas d'une attaque, le calcul de l'empreinte afin d'identifier le flux est exécuté sur tous les paquets entrants. Le calcul est inutile mais constant, et ne cause donc pas de problème de charge, contrairement à la mémoire occupée. En effet, l'insertion d'une empreinte en mémoire suite à la réception d'un premier paquet SYN, vient utiliser de l'espace mémoire et pourrait forcer des évictions des tables selon l'ampleur de l'attaque.

Afin de remédier à cela, il peut être avantageux de ne faire que des insertions de nouveaux

flux qu'une fois avoir reçu un second paquet depuis le client, mais reconnaître le flux nécessite d'écrire en mémoire. On pourrait alors décider d'un délai assez court pour l'éviction lorsque la communication n'est pas établie.

3.2.3 SIP

Le protocole SIP [24] (Session Initiation Protocol) est utilisé dans les appels sur IP (VoIP). Il permet d'établir, de modifier et de terminer une communication multimédia en temps réel basé sur IP. Le protocole se situe au niveau de la couche de session du modèle OSI. Ainsi, il nécessite également la présence d'un protocole de transport comme UDP ou TCP. Le protocole implique plusieurs nœuds réseau à savoir des agents utilisateur (client), un serveur mandataire, un serveur de registre et d'autres intermédiaires. La communication s'amorce en contactant le serveur mandataire qui agit comme intermédiaire avec l'agent utilisateur du destinataire. Une fois la réponse du destinataire obtenue, le mandataire contactera l'agent utilisateur qui a amorcé la connexion avec les informations du destinataire. Ensuite, une connexion s'établit entre l'initiateur et le destinataire. Cette connexion durera tout au long de l'appel afin de transporter la voix des usagés. Le serveur mandataire protège l'adresse réseau du destinataire jusqu'à ce que celui-ci accepte l'appel. Comme les requêtes au serveur mandataire sont beaucoup moins volumineuses et moins fréquentes que les échanges de paquets vocaux, il s'agit là d'un exemple parfait de la distinction entre le plan des données et le plan de contrôle. La signalisation vers le serveur mandataire n'a en effet pas les contraintes de temps réel que la voix nécessite, et donc est plus tolérant aux variations de délais de traitement. La gestion d'états est donc répartie entre les différents intermédiaires et les agents utilisateurs.

3.2.4 GTP

Le GPRS tunneling protocol (GTP) [25] sert à établir une connexion par paquet sur les réseaux cellulaires(GSM, UMTS, LTE et 5G). Il tire son origine des réseaux GSM, mais il continue d'être utilisé dans les réseaux 5G. Il existe deux protocoles dans la norme à savoir un protocole pour le contrôle GTP-C et un protocole pour les données GTP-U. Ces deux protocoles assurent la communication entre les *GPRS support nodes* (GSN). Ainsi, un tunnel établi entre deux GSN pour servir une connexion par paquet d'un appareil cellulaire peut se déplacer si l'appareil change le GSN qui le sert (*serving GSN* ou SGSN). GTP se base sur le protocole TCP et permet de fournir un tunnel IP dans le réseau cœur cellulaire. Selon ce protocole, on peut créer, modifier et supprimer un tunnel. On peut voir ainsi qu'il y a différents états de connexion qui interviennent, par exemple la communication TCP sous-

jacente ou simplement l'établissement ou non du tunnel.

3.2.5 Répartiteur de charge

Les répartiteurs de charge servent à diviser le trafic parmi un ensemble de serveurs. Le répartiteur permet d'exposer un ensemble de serveurs comme une seule ressource. Ainsi, on peut s'adapter à l'achalandage et ne pas surcharger les serveurs. Pour le répartiteur, la liste des serveurs cibles est un état global qui affecte chaque flux. Une fonction de hachage basée sur les valeurs identifiant le flux permet de s'assurer que le flux est acheminé au serveur correspondant. Pour les besoins du service qu'expose le répartiteur, chaque flux doit trouver le même serveur de manière a préserver la cohérence. C'est-à-dire que des requêtes répétées provenant d'un client en particulier seront toujours acheminées au même serveur. De plus, la répartition doit être bien équilibrée. Ces deux contraintes posent problème lors des modifications à la liste de serveurs en cas de panne ou d'ajout d'un nouveau serveur. En effet, en cas de panne, le trafic vers le serveur défaillant doit être réacheminé vers des serveurs fonctionnels. En cas d'ajout d'un nouveau serveur, il faut que les connexions existantes restent dirigées vers leurs serveurs correspondants, mais il faut également que les nouvelles connexions soient réparties sur l'ensemble modifié. Pour satisfaire les contraintes, l'évaluation de la fonction de hachage peut être effectuée une seule fois en début de connexion. Toutefois, il faut alors conserver en mémoire le résultat de la fonction de hachage, ce qui limite le nombre de connexions qu'un répartiteur peut traiter [26].

3.2.6 Pare-feu

Les pare-feux filtrent le trafic afin de protéger une portion du réseau des paquets indésirables. Un ensemble de règles provenant du plan de contrôle dicte comment identifier ces paquets. Les règles avec états sont des règles qui s'ajoutent et se retirent automatiquement en fonction de certains événements, par exemple si un ordinateur situé à l'intérieur d'un réseau protégé par un pare-feu effectue une requête vers l'extérieur du réseau. Afin de recevoir la réponse du serveur de l'extérieur du réseau, le pare-feu analysera la requête à sa sortie et ajoutera une règle pour permettre la réponse. À la fin de la communication, le pare-feu pourra retirer la règle. Les règles avec états renforcent beaucoup la sécurité en diminuant la fenêtre d'opportunité pour un attaquant. Aussi, ces règles s'adaptent automatiquement en fonction du trafic légitime et diminuent l'intervention des opérateurs.

3.2.7 Comparatif

Le tableau 3.1 présente un portrait global des applications analysées. L'identification fait référence au minimum d'information nécessaire pour isoler un flux particulier pour cette application. Le nombre d'états rapporte le nombres d'états possibles de l'application ou du protocole. La fréquence de mise à jour fait référence à la fréquence de modifications d'états en mémoire pour les données analysées. Les protocoles associés sont des protocoles qui peuvent être utilisés conjointement avec l'application ou le protocole en question.

Tableau 3.1 Tableau comparatif des différents protocoles et applications à l'étude.

Application	Identification	Nombre d'états	Fréquence de mise à jour	Protocoles associés
TCP	5-tuple	12	Forte	IP
Tempête de SYN	5-tuple	2 (12 pour TCP)	Forte	IP, TCP
SIP	5-tuple	4	Faible	IP, TCP, UDP
GTP	5-tuple	2	Moyenne	IP, TCP
Répartiteur de charge	5-tuple avec source partielle	2	Forte (selon la charge)	IP, TCP, UDP, autres
Pare-feu	5-tuple	2	Moyenne	IP, TCP, UDP, autres

L'identification peut en général être effectuée avec un 5-tuple qui est le plus sûr moyen d'identifier la provenance d'un paquet et son appartenance à un flux spécifique. Dans le cas d'un répartiteur de charge, il se peut que l'on répartisse la charge en fonction de l'adresse source et ainsi on peut employer un masque de sous-réseau. Ainsi, il n'est pas nécessaire d'identifier l'adresse source de manière exacte.

Le nombre d'états peut varier, mais généralement l'utilisation de deux états fait référence à la présence ou non d'une entrée dans une table pour ce flux. Par exemple, dans le cas d'un pare-feu, un flux peut soit être présent dans la table ou ne pas être présent et le traitement du flux s'en trouve affecté. Pour le nombre d'états de SIP, il existe plusieurs types de transactions et chacune de ces transactions peut être impliquée dans le signalement d'une connexion. Toutefois, les principales transactions utilisent une machine à quatre états.

3.3 Conclusion

Ce chapitre a couvert plusieurs concepts et applications concernant la gestion d'états. La distinction entre les flux et les sessions permet de comprendre que différents flux pouvant

avoir différents états peuvent être regroupés dans le contexte d'une application. La distinction entre les niveaux d'abstraction dans le réseau permet de comprendre que la gestion d'états peut nécessiter de considérer plusieurs composants réseau comme un seul système, tel que dans le cas des réseaux 5G. De plus, certaines valeurs propres à une communication peuvent aussi être considérées comme des états, par exemple le suivi des numéros de séquence de TCP. Parmi les applications couvertes, TCP nécessite une attention particulière. En effet, ce protocole est également utilisé comme couche de transport dans les applications comme la prévention de tempête de SYN, la VoIP avec SIP et les réseaux cellulaires avec GTP, sans compter les répartiteurs de charge et les pare-feux qui prennent en charges une multitude de protocoles différents. C'est donc sans surprise que TCP figure au centre du prochain chapitre.

CHAPITRE 4 ANALYSE DE FLUX TCP

Afin de permettre une bonne allocation des ressources du réseau, la catégorisation des flux est primordiale. Pouvoir déterminer à l'avance la durée de présence en mémoire ou la fréquence de nouvelles connexions permettrait de dimensionner correctement la mémoire et de choisir sur quel type de mémoire conserver l'information. La fréquence de changement d'états dicte la fréquence d'écriture et de lecture en mémoire. La durée d'un flux dicte leur durées de présence en mémoire. Or, il n'est malheureusement pas possible de prédire exactement l'avenir. Il est toutefois possible de déterminer des caractéristiques statistiques qui permettent d'estimer la probabilité d'appartenir à une certaine catégorie. Cette section porte sur la caractérisation des flux TCP dans un réseau et comment mettre à profit ces caractéristiques afin d'évaluer les contraintes sur la hiérarchie de mémoire. Elle est divisée en deux sous-sections : la méthodologie d'analyse et la caractérisation des flux.

4.1 Méthodologie d'analyse

Cette section porte sur l'analyse d'une trace prise en 2019 sur le réseau dorsal entre New-York et Sao-Polo [27]. Plus spécifiquement sur les paquets utilisant le protocole à états TCP.

4.1.1 Méthode de traitement

Analyser une trace et en faire le traitement dans le réseau partagent des problèmes communs, à savoir, la gestion de la mémoire dans le temps et l'aspect séquentiel de l'information. Une trace est une séquence de tous les paquets ayant traversé en point du réseau. Ainsi, pour faire l'analyse d'un flux, il faut minimalement retenir son 5-tuple en mémoire afin d'identifier ce flux et reconnaître quels autres paquets appartiennent à ce flux. Or, dans la trace qui a été utilisée, il y a des millions de flux différents et donc il faut faire attention à l'information conservée lors du traitement. Aussi, comme il s'agit d'une séquence d'information, l'ordre a son importance et cela peut limiter les possibilités de parallélisation des calculs. Pour remédier à ce problème, il faut pouvoir isoler les flux les uns des autres et les traiter indépendamment. Chaque flux peut ainsi être traité en parallèle. Toutefois, les flux varient tant en nombre de paquets qu'en durée et demandent donc des ressources non uniformes. En effet, lors du traitement, plus un flux dure longtemps, plus il faut conserver ses informations en mémoire. C'est également vrai pour l'analyse statique, mais ce n'est pas une problématique dans ce contexte. À l'opposé, la quantité de paquets par flux et la quantité de flux ont

un impact direct sur le nombre de calculs ainsi que sur la quantité de mémoire nécessaire, autant dans le réseau que dans l'analyse.

La trace est séparée en deux directions correspondant aux deux interfaces capturées. Chaque direction ayant une horloge de haute précision qui lui est propre, un certain décalage initial entre les deux directions est présent dans la trace. Les estampilles de temps de chaque paquet sont conservées séquentiellement dans un fichier séparé du fichier de trace afin de permettre une plus grande précision que celle permise par le format Pcap. La trace totale dure 1 heure, chaque fichier de trace porte sur environ une minute pour une seule direction. Cela représente environ 120 fichiers de 2 à 6 Go accompagnés de 120 fichiers d'estampilles de temps. Comme chaque fichier est borné en temps, la taille de ce fichier est une fonction du débit d'entête. Les interfaces utilisées pour chaque direction prennent en charge 10 Gb/s, mais l'utilisation de chaque lien est de $45,1\%$ et $47,1\%$. Toutefois, le taux de paquets transmis par seconde varie beaucoup, à savoir 635 mille paquets/s et 1.56 Millions de paquets/s. Le nombre de paquets transmis par seconde étant différent selon chaque direction, la taille moyenne d'un fichier est de 2.4 Go pour la direction de São Paulo vers New York et de 6 Go pour l'opposé.

Il est à noter que la partition arbitraire de la trace en minutes divise les flux qui ont des paquets dans deux fichiers ou plus. Tous les flux ayant une durée de plus d'une minute se trouvent également partitionnés dans plusieurs fichiers. Par ailleurs, la durée d'un flux n'est pas toujours claire. Il se peut que les flux en début de trace aient existé avant la prise de la capture, similairement pour les flux de fin de trace. Pourtant, même pour les flux entièrement captés, il est parfois difficile de déterminer la fin d'un flux puisqu'il se peut que le flux ne se termine pas explicitement. En effet, TCP a un signal de fin de communication qui n'est pas toujours employé. Bon nombre de communications TCP se terminent après une durée prédéterminée, durée qui tourne autour de 5 minutes. Donc, il faut lire plusieurs fichiers pour garantir la fin d'un flux.

Une caractéristique intéressante des flux est la distribution de leur durée. En effet, les flux courts sont de plusieurs ordres de grandeur plus nombreux que les flux les plus longs (voir la section 4.2.2). La grande quantité de flux courts peut poser problème si l'on accumule de l'information en fonction du nombre de flux. Par exemple, si on utilise une table de hachage pour répertorier chaque flux, une grande quantité de flux n'auront qu'un ou deux paquets, mais vont quand même utiliser la mémoire pour un flux.

La première étape de traitement est donc de séparer les flux les plus longs des flux les plus courts et de se débarrasser d'un partitionnement en temps pour un partitionnement en flux. Pour connaître la durée du flux, il faut savoir quand il débute et quand il se termine. Le début d'un flux doit se trouver dans la trace pour être considéré. En effet, il est impossible d'établir la durée d'un flux ayant commencé avant la capture. Il en est de même pour la fin du flux si elle se trouve passé la fin de la capture. Toutefois, tel que mentionné précédemment, les flux peuvent se terminer après une durée déterminée. Cela permet d'établir une fenêtre de temps pour trouver un paquet appartenant au flux. Comme le partitionnement des fichiers est en temps, on peut ainsi borner la recherche du dernier paquet de la connexion dans les cinq fichiers suivant le fichier à l'étude. Il faut cinq fichiers pour garantir que le délai de cinq minutes soit entièrement contenu dans ces fichiers.

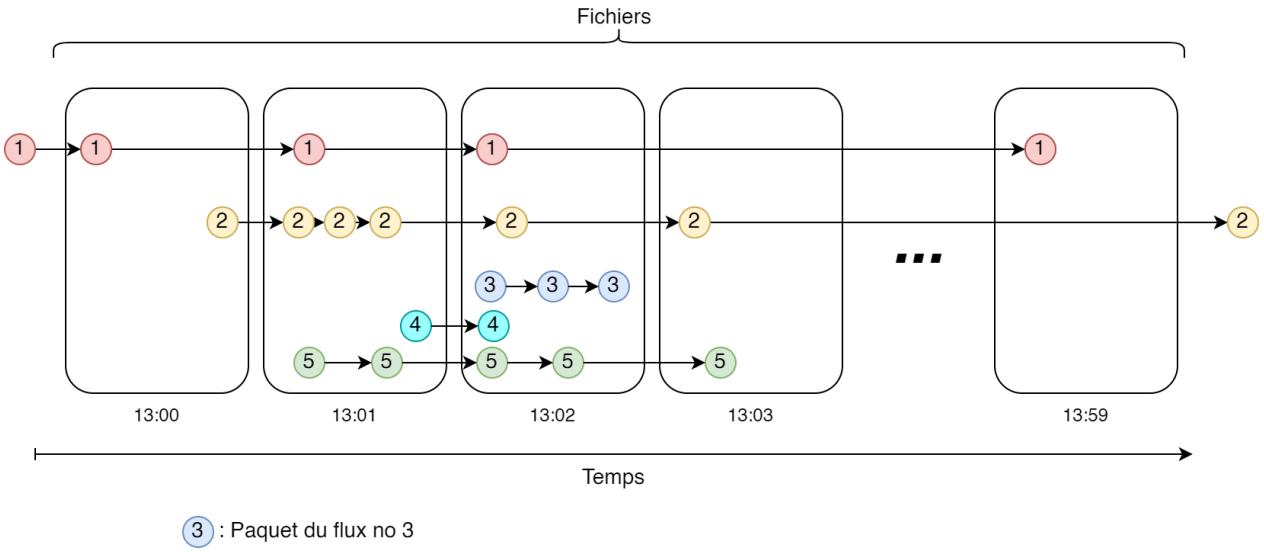


Figure 4.1 Le partitionnement en temps de la trace à l'étude.

La figure 4.1 montre une répartition possible des paquets de cinq flux sur les 60 fichiers d'une capture entre 13 h et 14 h. Le flux **1** commence avant la capture et sera donc rejeté, puisque ses caractéristiques ne seront pas représentatives. En effet, il est impossible de récupérer l'information manquante. Le flux **2** commence dans la trace donc il sera considéré. Toutefois, le dernier paquet du flux se trouve en dehors de la trace.

Il y a trois cas de figure ici. Un flux peut ne pas avoir de terminaison ou se terminer implicitement ou explicitement. Un flux n'a pas de terminaison s'il est impossible de savoir si

le dernier paquet de la capture pour ce flux est le dernier paquet de la communication. Ce cas arrive surtout si le dernier paquet du flux se trouve dans les cinq dernières minutes de la capture. Si le paquet ne termine pas la connexion explicitement avec un fanion FIN, il faut pouvoir déterminer si la connexion se termine implicitement. Une connexion se termine implicitement si le délai de terminaison est atteint et qu'aucun autre paquet appartenant à ce flux n'est reçu durant cette période. En d'autres termes, pour un flux ayant un paquet dans les cinq dernières minutes de la capture, il est impossible de garantir qu'un paquet arrive ou non, avant la fin du délai de terminaison. En effet, la capture se termine avant la fin du délai de terminaison.

Le flux **3** est entièrement contenu dans un seul fichier. C'est le genre de flux que l'on considère comme étant court. Bien qu'il se pourrait qu'il y ait des flux plus courts qui chevauchent deux fichiers, comme le flux **4**, ceux-ci sont considérés comme longs pour les fins du partitionnement en flux. En fait, pour les besoins du traitement, ce qui nous importe c'est si un flux possède des paquets dans plus d'un fichier de la capture. Afin d'analyser un flux comme le flux **4**, il faut lire tous les fichiers contenant un paquet appartenant au flux. Or, en regroupant tous les paquets appartenant à ce flux dans un seul fichier, il suffit de lire ce fichier pour analyser le flux. De plus, comme les flux sont indépendants les uns des autres, il est possible d'effectuer l'analyse des flux en parallèle. Chaque fichier pourra ainsi être lu simultanément, ce qui n'est pas possible avec un partitionnement en temps.

La figure 4.2 montre comment les flux **3**, **4** et **5** seraient répartis en fichiers selon le partitionnement en flux. Les encadrés représentent les fichiers, les différents blocs représentent les données d'un paquet.

Le fichier nommé «Flux du fichier 13 :02» contient tous les flux qui sont entièrement contenus dans le fichier de la capture couvrant la minute 13 : 02 comme on peut l'observer à la figure 4.1. Toutefois, les paquets des flux **4** et **5** ne s'y retrouvent pas, bien que ces flux ont des paquets qui se trouvent dans le fichier correspondant à cette minute. Or, ils ont également des paquets présents dans d'autres fichiers de capture, notamment le fichier de la minute précédente 13 : 01 qui comporte les premiers paquets de ces deux flux. Si l'on avait conservé le partitionnement en temps, pour traiter le flux **5** il faut lire séquentiellement les fichiers pour obtenir une séquence de délais entre les paquets.

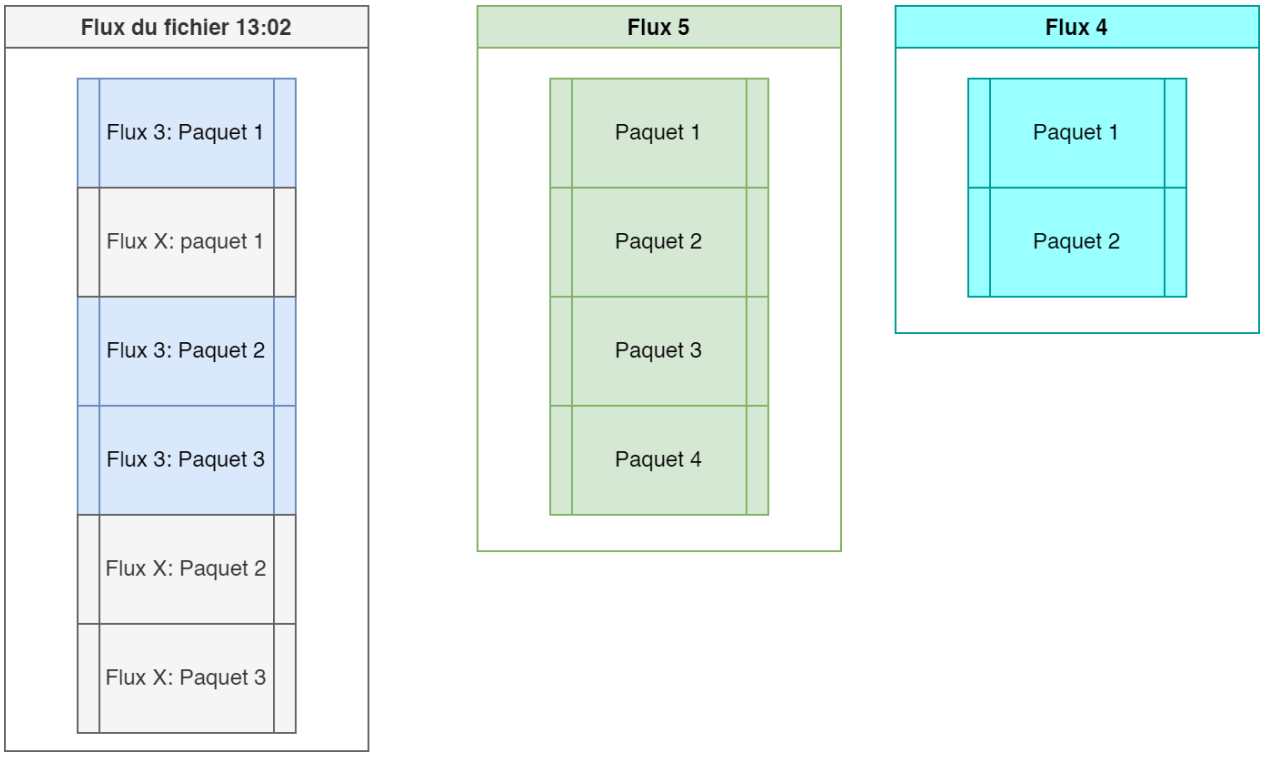


Figure 4.2 Le partitionnement en flux de la trace à l'étude

4.1.2 Outils de traitement

Le traitement s'effectue au moyen de deux programmes. Le premier sert à manipuler des ensembles de 5-tuples pour identifier les flux dans plus d'un fichier et le second pour obtenir des métriques sur ces flux et les inscrire dans un fichier CSV. Toutefois, comme les fichiers de trace sont au format PCAP avec des estampilles de temps séparé il faut d'abord extraire l'information nécessaire. En effet, ce format contient la séquence de paquets capturés, tous protocoles confondus. Cela signifie qu'il faut analyser le contenu du paquet pour déterminer sa taille et ainsi pouvoir lire le paquet suivant. Ainsi, en prenant uniquement les paquets TCP et en sélectionnant uniquement l'information nécessaire on peut réduire la taille des fichiers à traiter. Il est également possible de lire un emplacement arbitraire dans le fichier puisque toutes les informations ont la même structure et la même taille. Chaque paquet est représenté par son 5-tuple, l'estampille de temps associée, ses fanions TCP, son numéro de séquence ainsi que la taille de sa fenêtre d'envoi. Donc pour chaque paquet TCP les informations sont réunies dans une structure de 72 octets.

Une fois les fichiers de prétraitement obtenus, il faut maintenant identifier les 5-tuples des flux se trouvant dans chaque fichier. En effet, l'ensemble des 5-tuples présent dans un fichier nous

permet de faire la distinction entre les flux qui sont dans un seul fichier et ceux qui sont dans plusieurs fichiers. Si l'on ne séparait pas ces flux, le traitement des flux pourrait manquer de mémoire disponible rapidement. En effet, la structure de données utilisée pour enregistrer les flux est une table de hachage basé sur un arbre binaire de recherche. Cette table comporterait minimalement une entrée par 5-tuples distinct. Ainsi, les flux qui se termine dans le fichier, qui ne seront plus utiles dans le futur vont continuer d'occuper la mémoire pour le reste du traitement. Or les flux de petite taille sont entièrement contenus dans le fichier et donc en lisant uniquement ce fichier il est possible d'obtenir toutes les informations nécessaires sur ces flux. Ce n'est pas le cas des flux longs qui nécessite la lecture de plusieurs fichiers pour déterminer leur fin et pour obtenir tout leur paquet, tel que présenté à la section 4.1.1. Mais comment savoir si un flux se termine dans le fichier ou s'il sera actif dans le futur, c'est-à-dire qu'il a des paquets dans au moins un des 5 prochains fichiers correspondant aux 5 prochaines minutes ? Il suffit de construire des ensembles de 5-tuples distincts pour chaque fichier. À partir de ces ensembles, si l'on cherche les flux qui ont des paquets dans deux fichiers, il suffit de faire l'intersection des ensembles de 5-tuples de ces deux fichiers. Ainsi, l'on peut définir l'ensemble des flux actifs à la fin d'une certaine minute comme l'union des intersections entre ce fichier et chaque fichier correspondant au 5 minutes suivant le fichier à l'étude. Tout 5-tuple se trouvant dans cet ensemble appartient donc à un flux actif à la fin de la minute, car il existe au moins un paquet de ce flux qui arrive avant la fin du délai de terminaison. De plus, en prenant ces mêmes intersections, mais en faisant l'union des intersections des 5 minutes avant la minute à l'étude, on trouve les flux qui sont actifs avant la minute. Nous avons donc besoin de trouver 2 ensembles, les flux actifs dans le future F et les flux actifs dans le passé P. Ces deux ensembles sont définit comme suit :

$$F_n = \bigcup_{m=n+1}^{n+5} S_n \cap S_m \tag{4.1}$$

$$P_n = \bigcup_{m=n-5}^{n-1} S_m \cap S_n \tag{4.2}$$

où n représente la minute à l'étude, S_n l'ensemble de 5-tuples à la minute n, F_n l'ensemble des flux présents dans le futur et P_n l'ensemble des flux actifs dans le passé. Il est intéressant de noter que l'intersection entre F_n et P_n existe et représente les flux qui ont un paquet dans la minute qui existait avant et qui continueront d'exister. Si l'on prend $F_n - P_n$ on obtient l'ensemble des 5-tuple de flux se terminant dans la minute. Si l'on prend $P_n - F_n$ on obtient l'ensemble des 5-tuples de flux qui commencent dans cette minute. En prenant $P_n \cup F_n$, on obtient l'ensemble des flux couvrant plus d'un fichier ayant un paquet contenu dans ce fichier. En utilisant cet ensemble, $P_n \cup F_n$, il nous est possible de filtrer les paquets et de les réunir dans

un fichier correspondant au flux. Le premier programme permet de générer les ensembles, effectuer une union, effectuer une intersection ou effectuer une soustraction d'ensemble de 5-tuples et d'utiliser un ensemble pour filtrer les paquets. De plus, les intersections utilisées pour trouver les flux actifs dans le futur peuvent être réutilisées pour trouver les flux passés. En effet, si l'on prend l'exemple de $S_0 \cap S_5$ on le retrouve dans l'union nous donnant F_0 ainsi que dans l'union donnant P_5. L'opération de filtrage permet de construire une structure de fichier où chaque fichier ne contient qu'un seul flux. Comme l'on ne veut pas de doublon dans les fichiers de flux, il faut être certains de filtrer une seule fois chaque fichier de prétraitement. L'ordre de filtrage n'a d'importance que si l'on veut éviter d'ordonner les paquets du flux en fonction de leurs estampilles de temps.

La structure de fichier est importante ici, puisque bon nombre d'opérations de recherche et de lecture de fichier dépendent du nombre de fichiers par dossier. Ainsi, plus le nombre de fichiers est grand, plus les opérations de traitement seront ralenties. La solution est donc de séparer les fichiers dans une arborescence de dossier. Afin de permettre une recherche rapide d'un flux en particulier, l'adresse de source est utilisée. Chaque octet de l'adresse correspond à un dossier dans l'arbre. L'adresse source a été choisie parce qu'elle est anonymisée ce qui garantit une certaine dispersion des adresses qui est augmentée par le processus semi-aléatoire d'anonymisation. En d'autres termes, le processus d'anonymisation augmente l'entropie des adresses. Cela permet de mieux répartir les fichiers de flux dans l'arborescence de fichiers tout en garantissant une opération de recherche de flux à partir du 5-tuple en temps constant. Il est possible de joindre les flux des deux directions en utilisant l'adresse source pour une direction et l'adresse de destination pour l'autre, ainsi le chemin de dossier sera le même dans les deux cas. Toutefois, les estampilles de temps entre les deux directions ont un décalage qui peut poser problème lors du calcul de métrique. Les deux directions ont donc été séparées pour obtenir les métriques présentées à la section 4.2. Comme chaque fichier contient un seul flux, on peut nommer le fichier avec son 5-tuple. Ainsi chaque fichier de flux ayant la même adresse source se retrouvera dans le même dossier, mais avec des noms distincts.

Une fois les flux organisés dans une arborescence de fichiers, il faut obtenir les métriques de chaque flux avec le second programme. Celui-ci peut lire en parallèle un maximum d'environ 2000 fichiers de flux simultanément en une dizaine de secondes et produire un fichier CSV avec une ligne de données par flux. Chaque fichier est lu par un fil d'exécution séparé qui remplit un tampon avec les informations de chaque paquet lu. Le tampon est lu par un autre fil d'exécution qui assigne un tampon pour chaque 5-tuple et qui remplit le tampon avec les informations de chaque paquet ayant ce 5-tuple. Un dernier fil d'exécution s'occupe seulement des paquets du flux et effectue les calculs des délais, la séparation des niveaux hiérarchiques et l'écriture de ces données dans un fichier CSV. Une fois que le fil d'exécution effectuant

la lecture du fichier de flux atteint la fin du fichier, un signal de terminaison est transmis par les différents tampons. Ainsi, avec le signal de fin de fichier il est possible de finaliser les calculs de métriques. Il serait possible de varier la taille des tampons pour optimiser le traitement. Il est important de noter qu'en traitant environ 2000 fichiers de flux il est fort probable d'atteindre la limite du nombre de fichiers ouverts pour un processus. En effet, il est possible que certains fichiers soient ouverts plus d'une fois et la limite peut être aussi basse que 1024 fichiers.

La principale limite de cette approche vient des accès au disque. En effet, les opérations de lecture et d'écriture dépassent de loin toutes les autres opérations. De plus, ces opérations requièrent souvent des appels au noyau du système d'exploitation, ce qui ralentit encore le traitement. L'opération de filtrage des paquets pour construire les fichiers de flux dans l'arborescence de fichiers est l'opération la plus longue. Toutefois, une fois cette opération faite, obtenir les métriques est assez rapide. D'ailleurs, il est possible d'analyser plusieurs fois cette arborescence pour obtenir n'importe quelle métrique de flux.

4.1.3 Hiérarchie de transition d'états

Lors d'une communication TCP, tous les paquets n'engendrent pas nécessairement une transition d'état. De plus, certains champs de l'en-tête TCP évoluent dans le temps selon la séquence de paquets. Les valeurs contenues dans ces champs peuvent modifier le traitement d'un paquet, on peut alors considérer ces valeurs comme des états de la communication. Cette section porte sur la distinction entre les différents types d'états et leurs relations.

Tout d'abord, rappelons que le traitement à états dépend du paquet entrant et de l'état mémorisé, tel que présenté à la section 3.1. Par exemple, considérons une opération qui calcule la somme cumulée de la taille des paquets. La valeur de la somme est conservée en mémoire. Le résultat du cumul dépend de la somme courante et du paquet entrant. Nous considérons ce type d'opérations comme du traitement à états, même si dans ce cas-ci l'ordre d'arrivée des paquets n'a pas d'importance, du fait de l'associativité de l'addition. Ensuite, dans le cas de TCP, plusieurs valeurs sont conservées en plus de l'état actuel de la connexion TCP, notamment le numéro de séquence et la taille de la fenêtre de réception. Le numéro de séquence est modifié lorsque la réception est confirmée, donc tous les paquets de réponses, fanion ACK, comportent une mise à jour du numéro de séquence. Tous les paquets suivant le paquet d'initialisation, fanion SYN, comportent une mise à jour du numéro de séquence. La taille de la fenêtre de réception, quant à elle, varie en fonction de l'espace disponible (en tampon) pour recevoir des paquets. La taille de la fenêtre peut également être modifiée

si l'on détecte de la congestion et que l'on gère cette congestion par réduction de fenêtre. Comme les modifications de la taille de la fenêtre sont transmises par un paquet d'accusé de réception ACK, il se trouve alors que le numéro de séquence est également mis à jour. Toutefois, ce ne sont pas tous les accusés de réception qui engendrent une modification de la taille de la fenêtre. On peut alors établir une relation d'ordre entre ces états. L'état de la taille de fenêtre a un impact sur la progression des numéros de séquence et des accusés de réception, par contre ceux-ci n'ont pas d'impact sur la taille de la fenêtre, seulement son avancement. De même, la progression de l'état de la connexion TCP s'accompagne toujours de la progression des numéros de séquence, certains états ont également un impact sur la taille de la fenêtre.

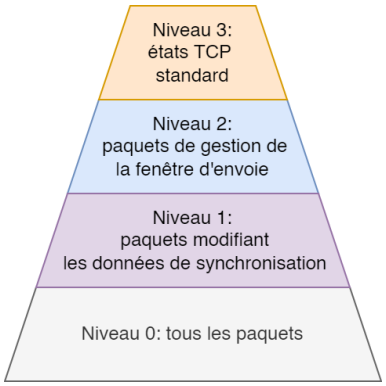


Figure 4.3 Hiérarchie d'états proposée pour le protocol TCP

Lors de l'analyse de la trace du CAIDA, nous avons déterminé 4 niveaux d'états, présentés à la figure 4.3. Les états à la base sont plus fréquents et sont parfois accompagnés de modifications d'états de niveau supérieur. Le niveau zéro est le plus bas ; tous les paquets reçus sont au moins de niveau 0. Le niveau 1 comporte tous les accusés de réception et toutes les modifications des numéros de synchronisation. Le niveau 2, quant à lui, considère surtout les modifications de la fenêtre d'envoi. Finalement, le niveau 3 comporte tous les paquets modifiant l'état de la connexion tel que présenté à la figure 3.2. Si l'on cherche à analyser les paquets d'un certain niveau, il faut donc considérer les paquets de niveaux supérieurs également. Il est important ici de faire une distinction entre la hiérarchie d'états propre à un protocole et le niveau d'abstraction d'état du réseau comme discuté à la section 3.1.1. Le niveau d'abstraction d'état permet de circonscrire le domaine d'application de cet état. Tandis que la hiérarchie d'états dans TCP s'applique uniquement au domaine d'une session TCP et donc sur les flux dans les deux directions.

4.2 Caractérisation de flux

Dans cette section nous présentons les résultats de l'analyse de la capture du CAIDA. Nous couvrons d'abord le nombre de flux actifs suivit par la distribution des durées de flux. Nous présentons ensuite la distribution du nombre de paquets selon la hiérarchie proposé à la section 4.1.3. Puis, nous présentons une distribution du taux de transfert ainsi que de la taille des flux. Pour finir, nous présentons la distribution des délais entre paquets et entre transition d'états.

4.2.1 Flux actifs

Le nombre de flux actifs simultanément est une métrique primordiale pour l'estimation de l'espace mémoire nécessaire. Toutefois obtenir cette métrique à partir d'une trace nécessite d'identifier les flux et de déterminé l'instant de début du flux ainsi que l'instant de fin. L'instant de début peut assez facilement être déterminé par l'arrivée du premier paquets du flux dans la trace, mais pour la terminaison c'est un peu plus complexe. En effet une majorité de flux se termine au bout d'un délai prédéfini sans aucun paquet de terminaison. Donc la fin de l'activité d'un flux dépend de ce délai. Donc pour déterminer qu'un paquet est le dernier d'un flux il faut regarder si c'est un paquet de terminaison ou si un autre paquet faisant partie du flux est présent avant la fin du délai de terminaison. Faire cela pour quelque flux est assez simple, mais le faire pour des millions de flux devient prohibitif. C'est pour cette raison que l'on utilise parfois une approximation basé sur la durée de la trace et le nombre de flux indépendant présent dans la trace. En divisant le nombre de flux par la durée de la trace on obtient un nombre moyen de flux actif. Scazzariello et coll. [8] ont effectué une regression linéaire basée sur l'approximation du nombre de flux actifs pour les traces du CAIDA de 2013 à 2019. Ils estiment le nombre de flux actifs à 200 milles flux plus 120 miles flux par Gb/s de trafic. Certaines traces ont toutefois des nombre de flux actifs moyens beaucoup plus élevé. Pour la trace que nous considérons, pour une seule direction, nous avons 4,72 Gb/s de trafic ce qui nous donnerais environ 766 milles flux actifs. Cette estimation est une borne inférieur. Toutefois, leurs estimations ne fait pas de distinction entre les flux TCP ou UDP. Or, si l'on analyse uniquement TCP, le nombre de flux actifs se situe quand même dans les millions ce qui n'est rien comparé au nombre de flux TCP distincts présent dans une minute de trace.

L'approche de séparation de fichier en fichier de flux nous permet de séparer les flux longs des flux courts. Toutefois, il est possible d'interprété les flux longs et les flux court différemment. Les flux courts sont basé sur l'ensemble des flux présent dans une trace d'une minute qui ne sont pas présent dans les prochaines 5 minutes. Les flux longs sont les flux présent dans cette

même trace qui sont présent dans les 5 prochaines minutes. Ainsi, le nombre de flux long représente le nombre de flux actifs après avoir accumulé une minutes de trafic. Le nombre de flux court représente le nombre de flux que l'on pourra évincé de la mémoire, puisqu'il ne seront pas utilisé avant la fin du délai de terminaison. Ainsi, ces deux chiffres nous montre un instantanée des flux actifs à la frontière entre deux minutes. La sommes de ces deux nombres nous donnes le nombre de flux distinct présent dans la minute. Ainsi, la sommes nous montre le nombre d'entrée mémoire qui seront introduit pendant la minute et le nombre de flux long le nombre de flux qu'il faut conservé en mémoire. Ainsi, si un flux se termine dans cette minute, le flux ne se retrouvera pas dans les prochaines minute et ne sera pas considéré actif pour cette minute.

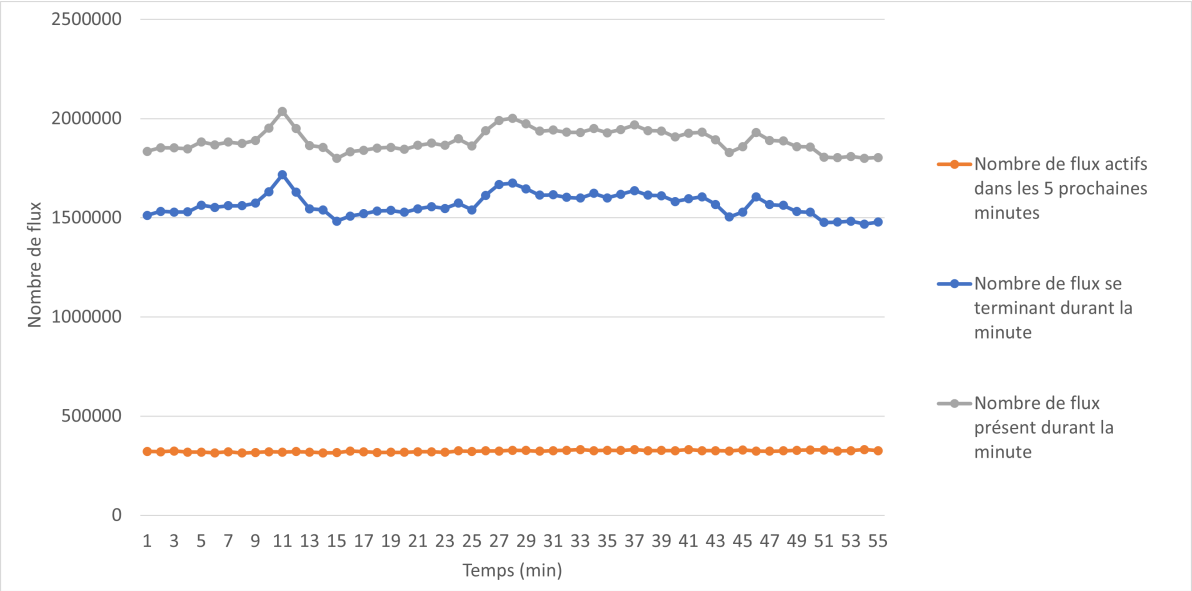


Figure 4.4 Activité des flux pour la direction A

La figure 4.4 nous montre la séquence d'instantanés pour la direction A. On remarque que le nombre de flux actifs est assez stable, en moyenne 323 milles flux avec un écart type de $1,4\%$. On remarque également que le nombre de flux à évincer est 5 fois plus important que le nombre de flux actif si bien que la proportion de flux actifs tourne autour de 17% du nombre de flux total pour une minute. Cela signifie qu'il faudrait évincer plus de 80% des flux arrivé dans la minute. De plus, les flux évincés seront remplacés au courant de la minute suivante. Ce haut taux de remplacement montre que les insertions sont très fréquentes.

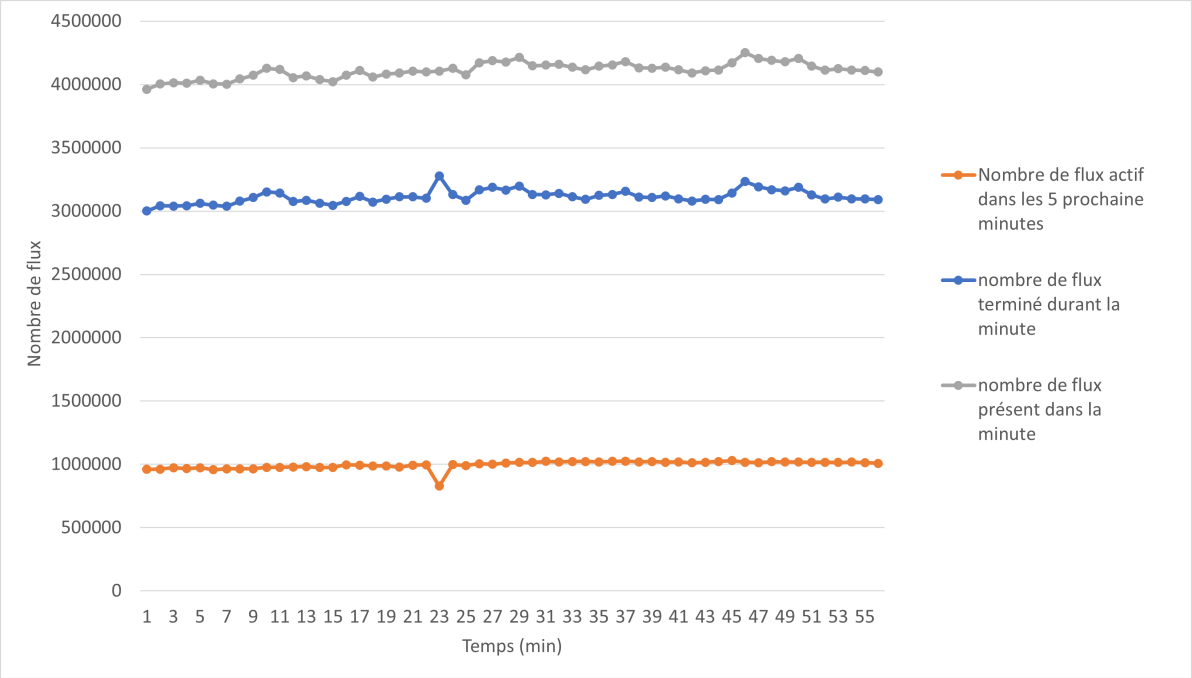


Figure 4.5 Activité des flux pour la direction B

La figure 4.5 nous montre la séquence d'instantanés pour la direction B. On remarque ici que le nombre de flux actif pour la direction B est plus important, en moyenne un million de flux. Aussi le nombre, de flux à évincé est le double du nombre obtenu pour la direction A, en moyenne 3,1 millions. L'écart type sur le nombre de flux actifs est plus grand que pour la direction A puisqu'il est de $3,2\%$. On peut supposer que la minute 23 pourrait être la cause de cette variation. On observe également que les flux actifs représentent en moyenne $24,3\%$ du nombre de flux présent dans la minute.

Une autre mesure intéressante pour la direction B est le nombre de long flux distinct présent dans la trace qui est de 25 millions. Le nombre de flux longs distinct nous offre également une idée approximative de la variation des flux actifs, même si cette valeur semble demeurée stable dans le temps. Ainsi en divisant le nombre de flux distinct par la durée de la trace on obtient un taux moyen de flux actifs de 7602 flux/s. Ce nombre nous donne une idée de la variation des flux actifs.

4.2.2 Durée de flux

La durée des flux est bornée à la durée de la capture considérée. Ici, comme nous nous bornons à 5 minutes, les plus longs flux ne peuvent dépasser cette valeur. La direction A va de São

Paulo vers New York et la direction B l'inverse. La figure 4.6 présente la distribution des durées de chaque flux pour la direction A.

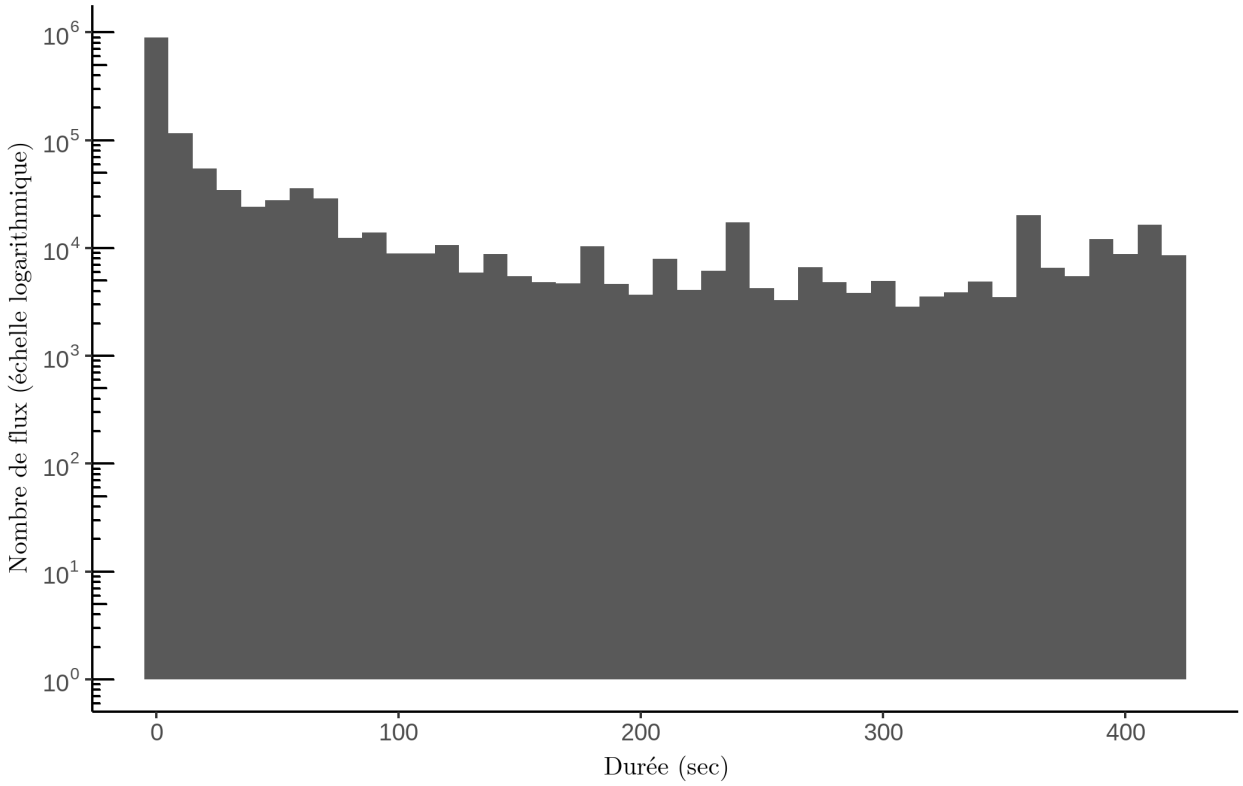


Figure 4.6 Distribution des durées pour la direction A

Étant donné, qu'il s'agit de flux long, le pique associé au flux court est moins prononcé. Toutefois, les flux longues durées sont deux ordres de grandeur moins fréquents que les flux très courts que l'on trouve proches de l'axe des y. La direction opposée est présentée à la figure 4.8

On peut constater qu'il y a légèrement plus de flux longs dans la direction B que dans la direction A. En effet, pour la direction B chaque bande de l'histogramme se retrouve au-delà de 10^4, ce qui n'est pas le cas de la direction A. La figure 4.7 montre la distribution des durées cumulées de chaque direction.

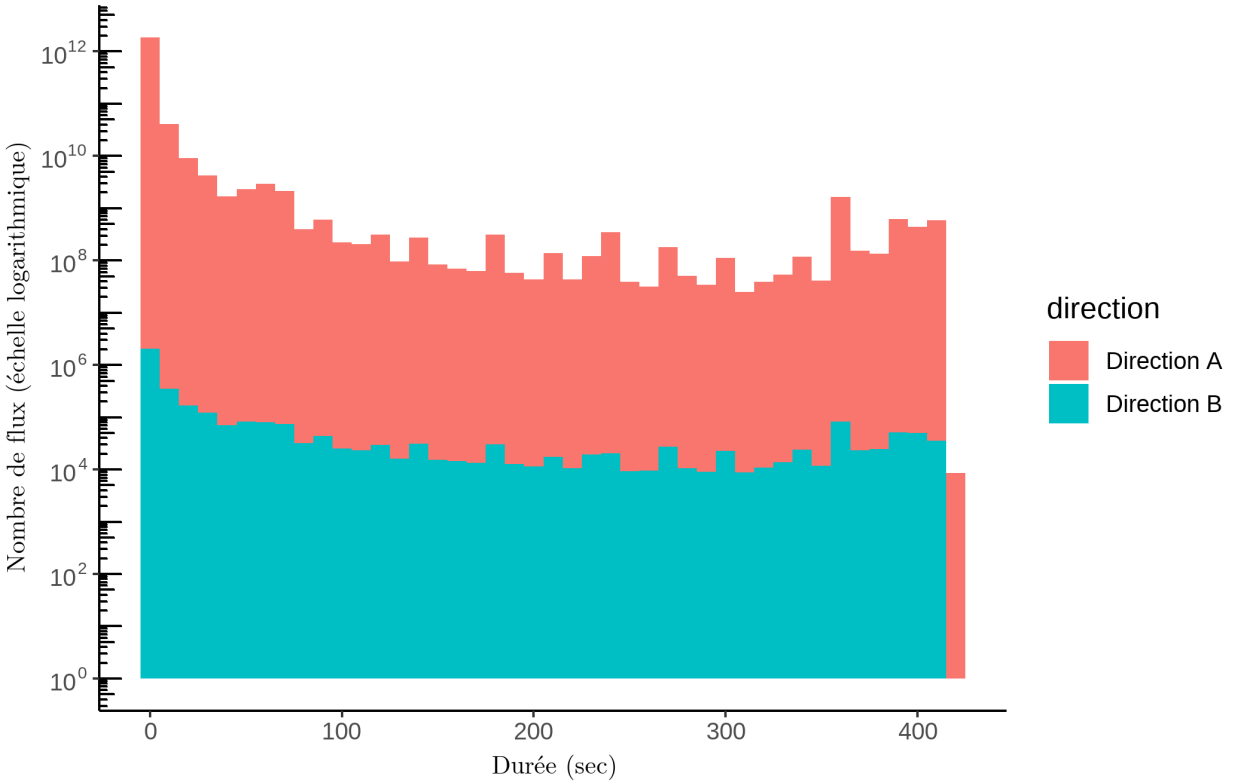


Figure 4.7 Distribution des durées pour les deux directions cumulées

Il apparaît évident en regardant l'ordre de grandeur de toutes les bandes de l'histogramme à la figure 4.7 que la quantité de flux pour les deux directions est proche de la centaine de millions. Cela montre l'intérêt de simplifier le traitement pour de tels flux. D'autant plus que cette durée signifie la durée d'occupation de la mémoire.

Si l'on regarde maintenant la distribution des durées de flux pour la direction B cumulé sur l'heure entière à la figure 4.8.

Il est intéressant de constater que bien qu'il y ait plus de définition, l'allure de l'histogramme est assez similaire avec l'histogramme pour les 5 premières minutes. Ce phénomène provient de l'aspect autosimilaire des traces réseau.

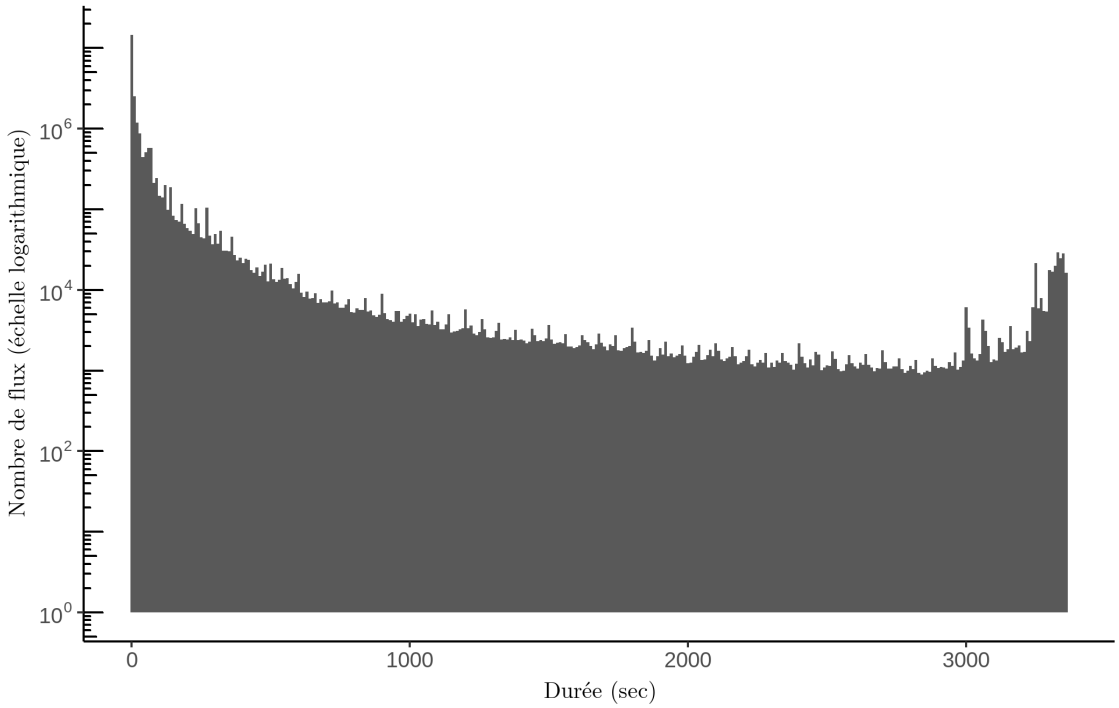


Figure 4.8 Distribution des durées pour la direction B

4.2.3 Hiérarchie d'états et nombre de paquets

Passons maintenant à l'analyse de la hiérarchie d'états. Chaque flux étant analysé indépendamment, le cumul de chaque niveau est fait pour chaque flux individuellement. Le niveau 0 couvre tous les paquets du flux et chaque niveau au-dessus réduit le nombre de paquets considérés. La figure 4.9 illustre la distribution du nombre de paquets par flux. On peut constater que les flux ayant beaucoup de paquets sont plusieurs ordres de grandeur moins fréquents que ceux qui ont un plus petit nombre de paquets. La figure 4.10 présente le total du nombre de paquets de niveaux 1 par flux. Rappelons que le niveau 1 signifie une modification des données de synchronisation, par exemple au moyen du fanion ACK. La figure 4.10 montre une diminution significative du nombre de paquets par flux. Effectivement, le fait de considérer uniquement les paquets de niveaux 1 réduit le nombre de paquets considéré. À la figure 4.9 on peut observer qu'il existe plusieurs flux de plus de vingt mille paquets. Toutefois, ces mêmes flux présentent moins de dix mille paquets de niveaux 1. On peut constater que la diminution du nombre de paquets continue avec les niveaux. La figure 4.11 présente la diminution du nombre de flux, mais certains flux comportent toujours le même nombre de paquets que pour le niveau 1.

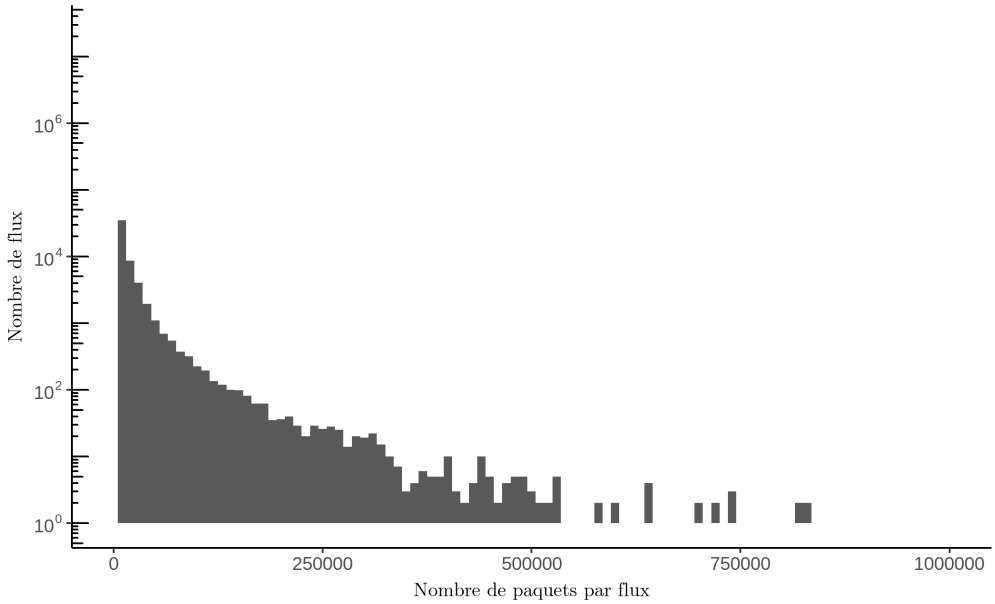


Figure 4.9 Distribution du nombre de paquets par flux pour la direction B

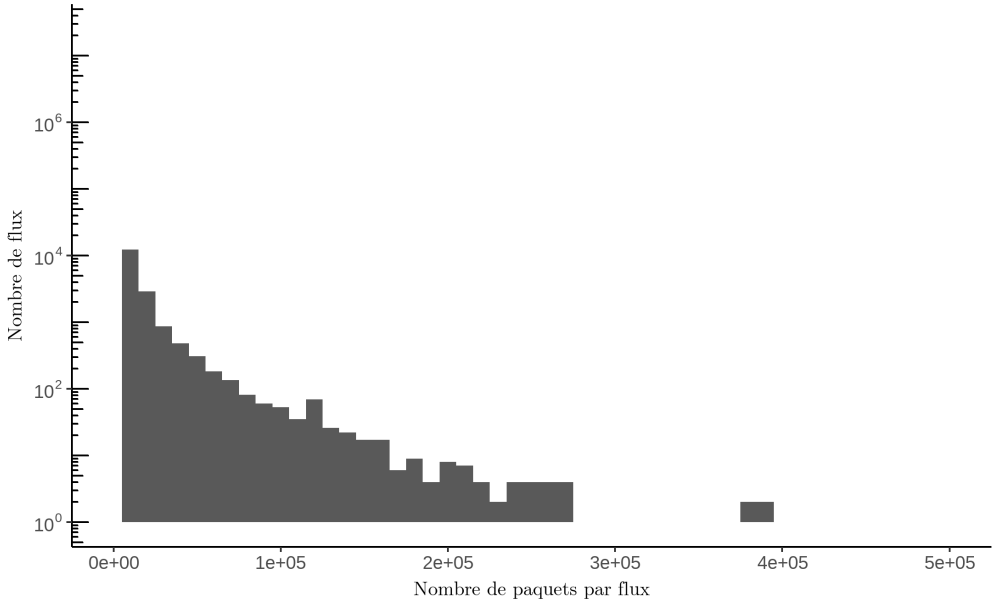


Figure 4.10 Distribution du nombre de paquets de niveau 1 par flux pour la direction B

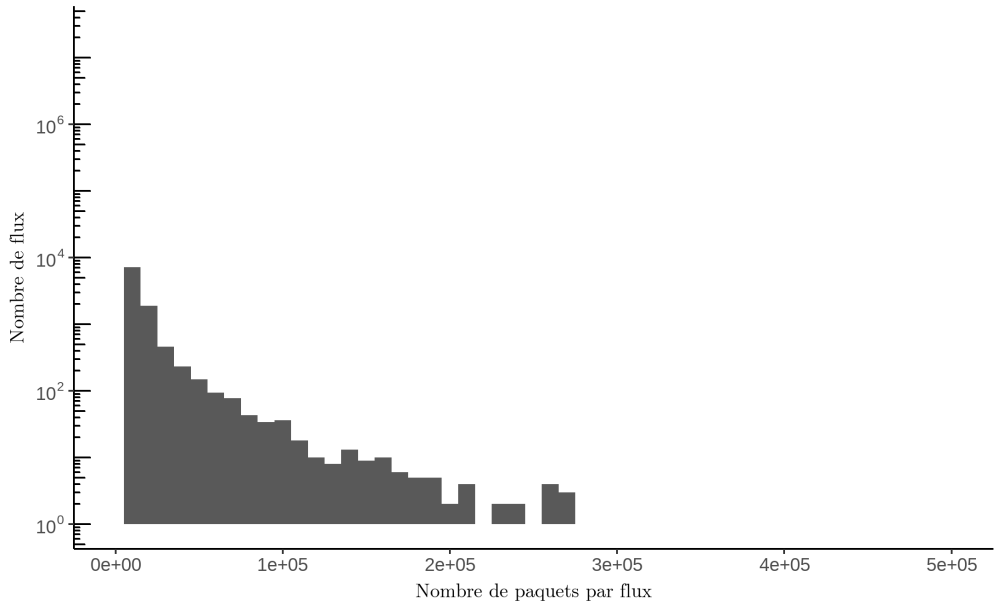


Figure 4.11 Distribution du nombre de paquets de niveau 2 par flux pour la direction B

4.2.4 Taux de transfert

Le taux de transfert représente le cumule de la tailles des paquets d'un flux d'un divisé par la durée du flux. Cette mesure nous informe sur la proportion de la bande passante utilisé pour un flux donné. La figure 4.12 présente la distribution de la somme de la taille de chaque paquets par flux.

On peut constater qu'avec l'échelle log-log une pente négative apparait dans la distribution. Cette pente signifie que plus les flux sont volumineux, moins il y en a. Cela pourrait s'expliquer par une répartition de la charge du trafic au niveau du réseau opérateur.

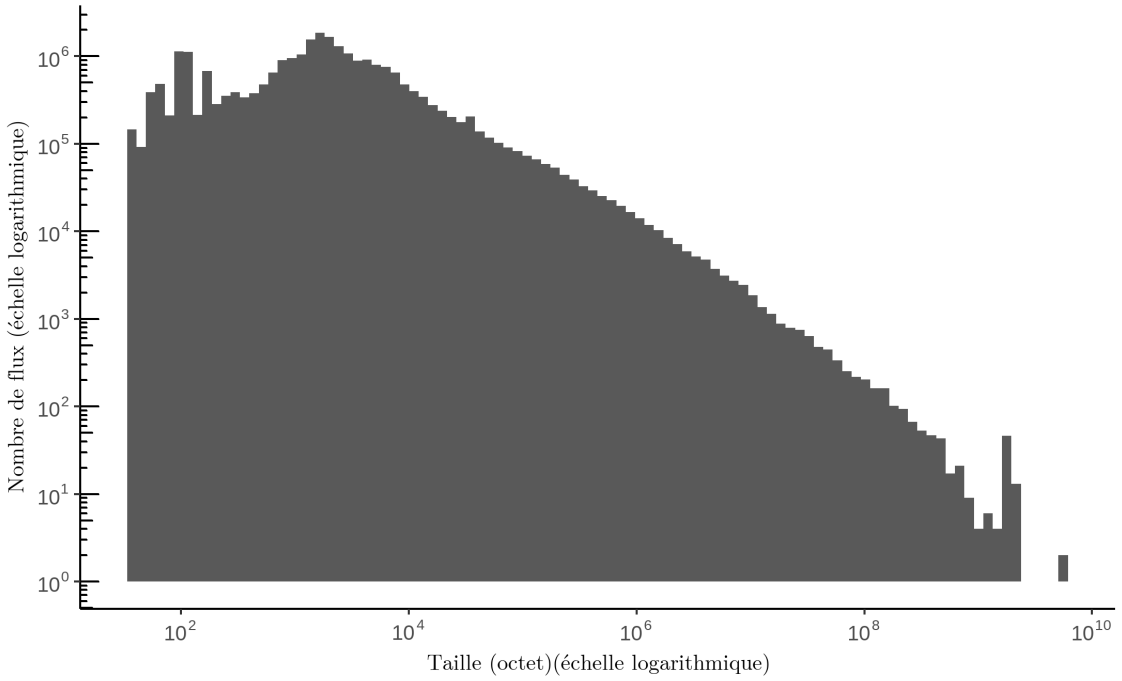


Figure 4.12 Distribution des taux de transferts cumulées pour les deux directions

4.2.5 Distribution des délais

Les délais entre l'arrivée de deux paquets d'un même flux est une borne pour le traitement d'états. En effet, la mise à jour de l'état courant doit pouvoir s'effectuer entre le traitement du premier paquet et l'arrivée du second. Or, cela reste vrai, peu importe le type d'état. Donc, en diminuant le nombre de paquets à considérer dans un flux et considérant un niveau plus élevé il est possible d'augmenter le délai entre les transitions d'états et donc de prolonger le temps disponible pour la mise à jour de l'état courant.

La figure 4.13 montre la distribution de la moyenne des délais entre tous les paquets consécutifs d'un même flux.

La figure 4.14 montre la distribution du délai médian entre les paquets d'un même flux. On peut constater que le délai médian a tendance à être plus élevé que le délai moyen. Il suffit de constater qu'il n'y a aucun flux présentant des délais moyens plus grands que 2000 secondes. Par contre, on peut voir à la figure 4.14 qu'il y a plus d'une cinquantaine de flux avec des délais médians de 3000 secondes.

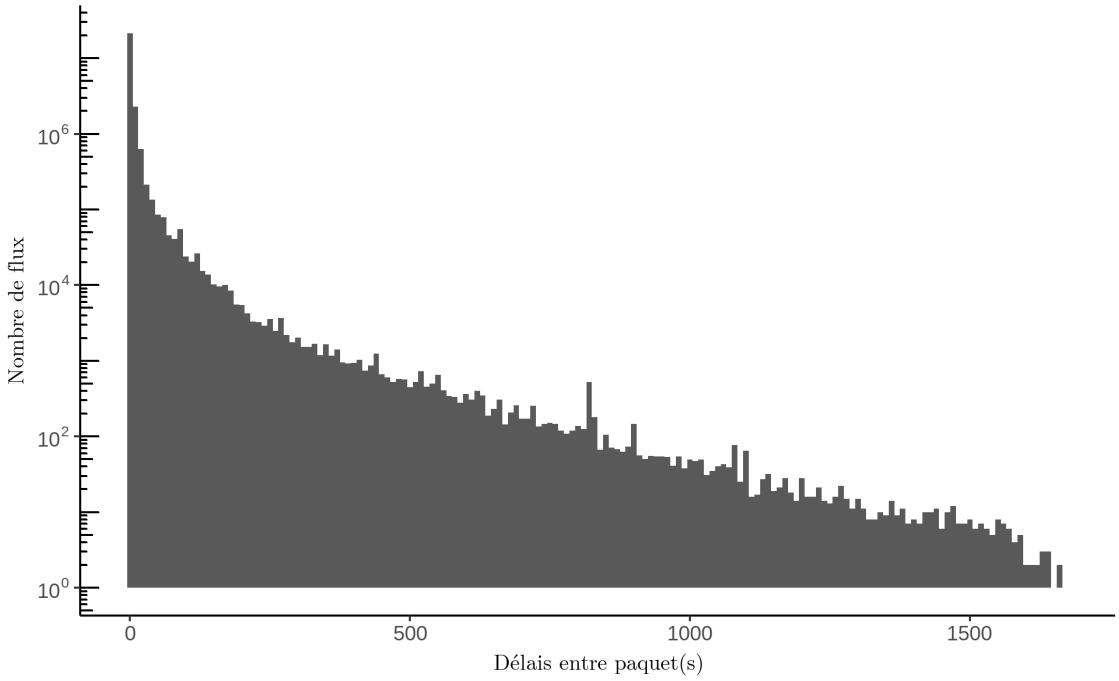


Figure 4.13 Distribution des délais moyens entre deux paquets par flux

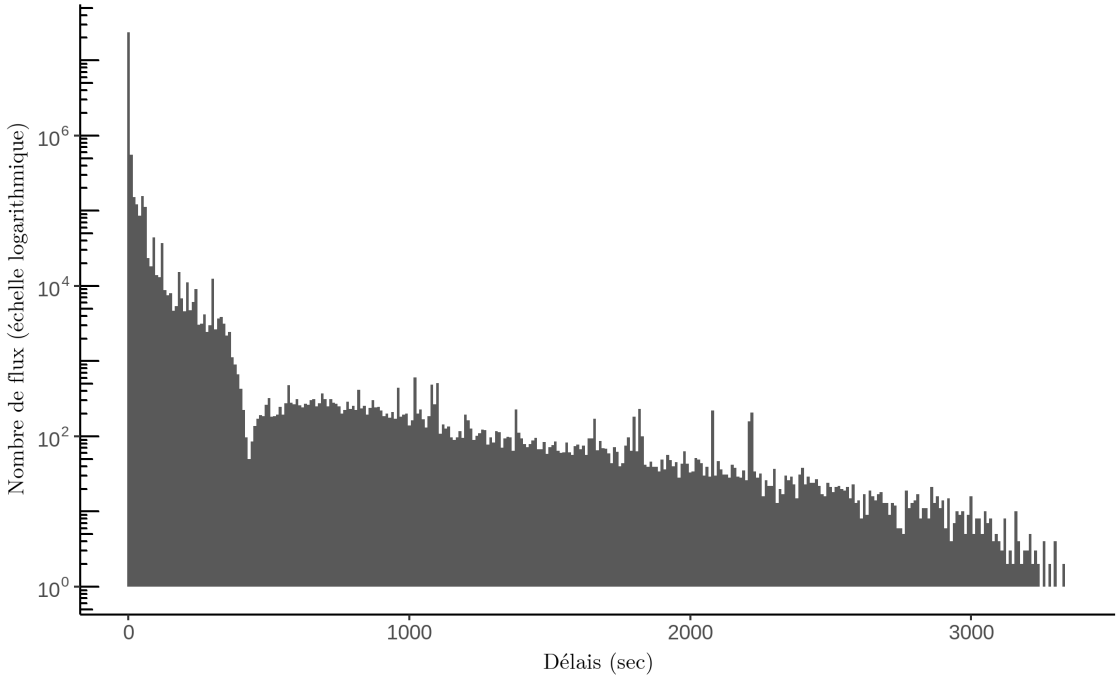


Figure 4.14 Distribution des délais médians entre deux paquets par flux

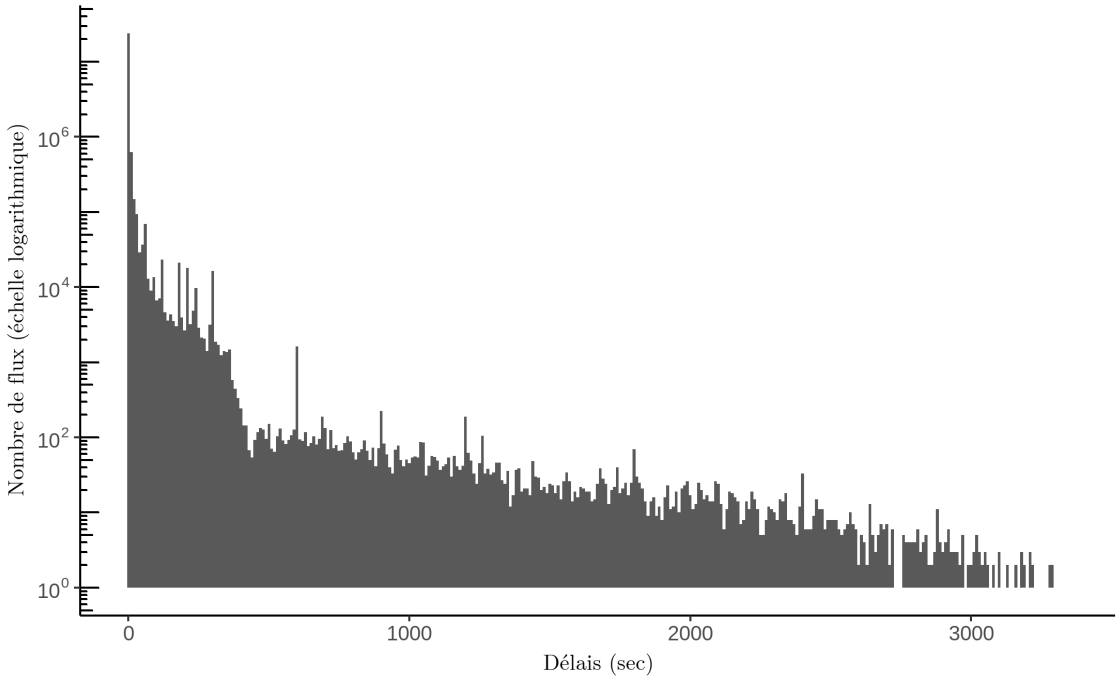


Figure 4.15 Distribution des délais médians entre deux paquets de niveau 1 par flux

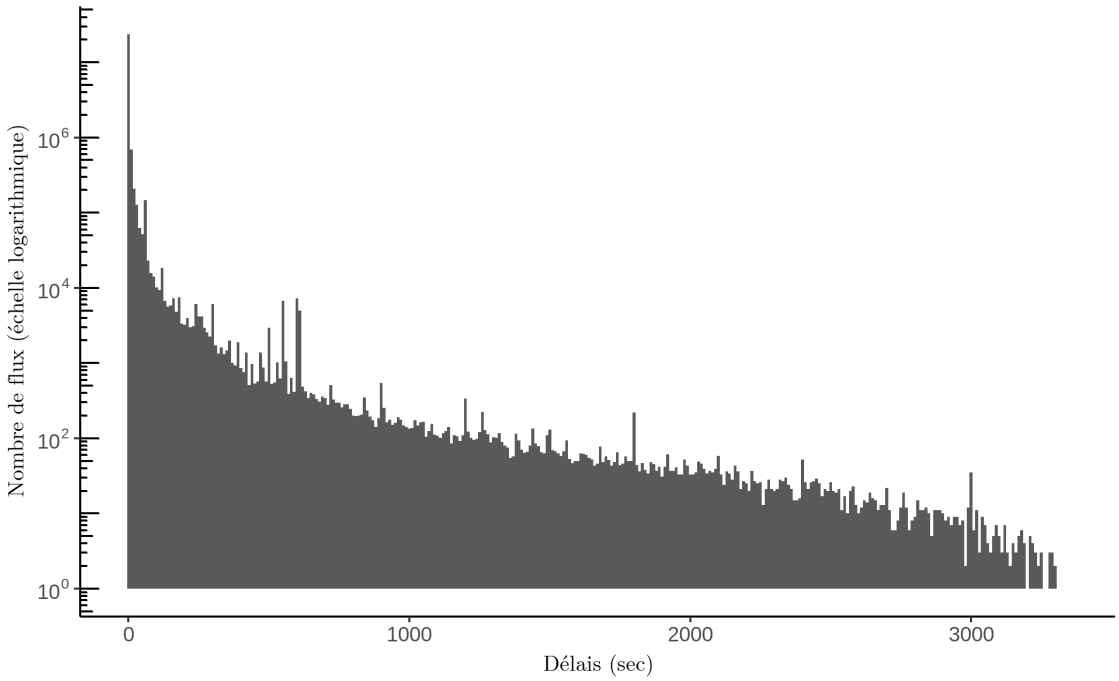


Figure 4.16 Distribution des délais médians entre deux paquets de niveau 2 par flux

On peut constater que plus on monte dans les niveaux, plus la distribution des délais médians s'aplatit. Plus l'on monte dans les niveaux, moins il y a de paquets considérés.

4.3 Discussion de la méthode de traitement

La méthode présentée permet d'obtenir diverses métriques à l'échelle du flux dans une trace volumineuse. Cette trace faisant plus de 500 Gb, une approche de traitement naïve ne peut tout simplement pas effectuer un traitement aussi fin. La trace du CAIDA n'a fait l'objet que d'une analyse sommaire du nombre de nouveaux flux en fonction du temps. Ce nombre est généralement une approximation basée sur le nombre de flux distincts divisé par la durée de la capture [8, 20]. Cette approximation est relativement bonne pour avoir une idée du nombre de nouveaux flux en fonction du temps, mais n'offre pas d'informations sur les délais entre les paquets dans un flux ni sur la durée de ces flux. La méthodologie utilisée dans ce mémoire permet d'avoir un portrait fin du comportement des flux. Sans ce portrait, il est difficile de comprendre les limites de la gestion d'états. Par ailleurs, la méthodologie présentée permet de faire une distinction à même le flux entre les différents paquets. Nous nous sommes intéressés au comportement du protocole TCP pour la gestion des états parce que c'est un protocole très utilisé et bien connu. Il est mature et performant, et profite à une majorité de connexions sur l'internet. D'ailleurs, la performance de TCP en fait un bon point de comparaison puisque les applications à états auront une performance égale ou inférieure. La prépondérance de ce protocole nous garantit également un échantillon considérable de comportements de flux, même si on ne considère que les flux longs. S'il n'est pas intéressant de distinguer les transitions d'états de TCP à l'intérieur du réseau, l'approche d'analyse hiérarchique reste néanmoins intéressante. En effet, cette analyse permet de traiter un sous-ensemble de paquets d'un flux basé sur d'autres caractéristiques que le 5-tuple. Si on a pu effectuer ce genre de traitement sur TCP dans une trace provenant d'une dorsale de l'internet, ce sera certainement possible d'effectuer ce traitement ailleurs et sur des traces plus courtes.

4.4 Conclusion

Dans ce chapitre, nous avons présenté une méthodologie de traitement des flux pour une capture volumineuse. Nous avons ensuite introduit le concept d'une analyse hiérarchique d'états. Puis nous avons présenté différents résultats concernant les variations entre les flux ainsi que la différence entre le traitement de différents niveaux d'états.

CHAPITRE 5 CONCLUSION

5.1 Sommaire des travaux réalisés et des résultats obtenus

Dans ce mémoire, nous avons abordé le problème de la gestion des états dans des réseaux informatiques à haut débit. Nous avons considéré en particulier les questions d'implémentation de cette gestion dans les commutateurs de ces réseaux.

Au chapitre 2, nous avons discuté de l'architecture des commutateurs programmables. Entre autres, nous avons abordé la séparation des plans réseaux, les fonctions d'un commutateur, les composants d'architecture d'un commutateur programmable ainsi que des langages spécifiques au traitement réseau qui accompagnent ces commutateurs. Nous avons également présenté les approches d'optimisation de la gestion d'états dans ces systèmes.

Au chapitre 3, nous avons proposé différents concepts basés sur des applications avec gestion d'états. Nous avons abordé différents concepts propres à la gestion d'états, notamment les niveaux d'abstraction d'états ainsi que la distinction entre un flux et une session. Nous avons également analysé 6 protocoles et applications avec gestion d'états.

Au chapitre 4, nous avons présenté l'analyse du comportement des flux TCP à partir d'une capture de trafic. Nous avons d'abord présenté une méthodologie d'analyse de flux accompagnée de deux logiciels permettant l'analyse à l'échelle des flux pour une trace réseau volumineuse. Nous avons ensuite décrit le comportement des flux TCP en termes du du nombre de flux actifs, de leur durée, du délai entre les événements et du débit.

5.2 Limite des contributions

Nous avons identifié les limites suivantes à notre travail. La trace que nous avons utilisée est limitée à un seul segment de l'internet. Il serait utile d'effectuer des captures en d'autres points de l'internet, dans des réseaux locaux ou des centres de données par exemple. La trace remonte à l'année 2019, et la nature et le débit du trafic pourraient avoir changé depuis. En particulier, l'augmentation substantielle du travail à distance suite à la pandémie a eu un effet sur la quantité de flux vidéo et audio. L'analyse des flux actifs présentés ne montre qu'une séquence d'instantanés. En conséquence, il peut exister des maxima ponctuels supérieurs à ce qui a été observé.

5.3 Recommandation pour travaux futurs

L'approche hiérarchique dans TCP pourrait être étendue aux protocoles de plus hauts niveaux dans la hiérarchie du modèle OSI. On pourrait compléter les analyses, à l'aide de la méthodologie proposée, pour des applications spécifiques telles que celles présentées au chapitre 3. Il est aussi possible d'étendre la méthodologie proposée afin de pouvoir effectuer ce traitement directement dans le réseau. En effet, il serait possible d'utiliser les commutateurs programmables pour effectuer l'analyse de traces volumineuses en s'inspirant de la méthodologie de traitement proposé. Il serait également intéressant d'explorer l'utilisation du processus stochastique Weibull qui semble être un outil approprié pour modéliser les flux [28].

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," *SIGCOMM Computer Communication Revue*, vol. 44, n°. 3, p. 87–95, jul 2014. [En ligne]. Disponible : https://doi.org/10.1145/2656877.2656890

[2] G. Gibb, G. Varghese, M. Horowitz et N. McKeown, "Design principles for packet parsers," dans *Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '13. IEEE Press, 2013, p. 13–24.

[3] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica et M. Horowitz, "Forwarding metamorphosis : Fast programmable match-action processing in hardware for sdn," dans *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA : ACM, 2013, p. 99–110.

[4] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown et S. Licking, "Packet Transactions : High-Level Programming for Line-Rate Switches," dans *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference - SIGCOMM '16*. Florianopolis, Brazil : ACM Press, 2016, p. 15–28.

[5] S. Pontarelli, R. Bifulco, M. Bonola, C. Cascone, M. Spaziani, V. Bruschi, D. Sanvito, G. Siracusano, A. Capone, M. Honda, F. Huici et G. Siracusano, "Flowblaze : Stateful packet processing in hardware," dans *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA : USENIX Association, févr. 2019, p. 531–548. [En ligne]. Disponible : https://www.usenix.org/conference/nsdi19/presentation/pontarelli

[6] P4-16 Portable Switch Architecture (PSA). [En ligne]. Disponible : https://p4.org/p4-spec/docs/PSA-v1.2.html

[7] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar et S. Seshan, "TEA : Enabling State-Intensive Network Functions on Programmable Switches," dans *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*. ACM, 2020, p. 90–106. [En ligne]. Disponible : https://dl.acm.org/doi/10.1145/3387514.3405855

[8] M. Scazzariello, T. Caiazzi, H. Ghasemirahni, T. Barbette, D. Kostić et M. Chiesa, "A High-Speed stateful packet processing approach for tbps programmable switches,"

dans *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA : USENIX Association, avr. 2023, p. 1237–1255. [En ligne]. Disponible : https://www.usenix.org/conference/nsdi23/presentation/scazzariello

[9] IETF, "Transmission Control Protocol," Internet Requests for Comments, Information Sciences Institute University of Southern California, RFC 793, Septembre 1981. [En ligne]. Disponible : https://tools.ietf.org/html/rfc793

[10] D. Patterson et J. Hennessy, *Computer Organization and Design RISC-V Edition : The Hardware Software Interface*, 2e éd. Cambridge, MA, USA : Morgan Kaufmann Publishers Inc., 2020.

[11] A. Sivaraman, N. McKeown, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall et S. Katti, "Programmable Packet Scheduling at Line Rate," dans *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference - SIGCOMM '16*. Florianopolis, Brazil : ACM Press, 2016, p. 44–57.

[12] N. Feamster, J. Rexford et E. Zegura, "The road to SDN : an intellectual history of programmable networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, p. 87–98, avr. 2014. [En ligne]. Disponible : http://dl.acm.org/citation.cfm?doid=2602204.2602219

[13] M. Shahbaz, S. Choi, B. Pfaff, C. Kim, N. Feamster, N. McKeown et J. Rexford, "PISCES : A Programmable, Protocol-Independent Software Switch," dans *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference - SIGCOMM '16*. Florianopolis, Brazil : ACM Press, 2016, p. 525–538.

[14] R. Giladi, *Network Processors : Architecture, Programming, and Implementation*, 1er éd. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2008.

[15] G. Bianchi, M. Bonola, A. Capone et C. Cascone, "OpenState : programming platform-independent stateful openflow applications inside the switch," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, p. 44–51, avr. 2014. [En ligne]. Disponible : https://doi.org/10.1145/2602204.2602211

[16] V. Shrivastav, "Stateful multi-pipelined programmable switches," dans *Proceedings of the ACM SIGCOMM 2022 Conference*. ACM, 2022, p. 663–676. [En ligne]. Disponible : https://dl.acm.org/doi/10.1145/3544216.3544269

[17] V. Gurevich. (2020) Programmable Data Plane at Terabit Speeds. [En ligne]. Disponible : https://opennetworking.org/wp-content/uploads/2020/12/p4_d2_2017_programmable_data_plane_at_terabit_speeds.pdf

[18] D. Moro, D. Sanvito et A. Capone, “FlowBlaze.p4 : A library for quick prototyping of stateful SDN applications in P4,” dans *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2020, p. 95–99.

[19] L. Zeno, D. R. K. Ports, J. Nelson et M. Silberstein, “SwiShmem : Distributed Shared State Abstractions for Programmable Switches,” dans *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, ser. HotNets ’20. Association for Computing Machinery, 2020, p. 160–167. [En ligne]. Disponible : https://doi.org/10.1145/3422604.3425946

[20] The CAIDA UCSD , “ Statistical information for the CAIDA Anonymized Internet Traces,” 2019. [En ligne]. Disponible : https://www.caida.org/data/passive/passive_trace_statistics

[21] C. Zeng, L. Luo, T. Zhang, Z. Wang, L. Li, W. Han, N. Chen, L. Wan, L. Liu, Z. Ding, X. Geng, T. Feng, F. Ning, K. Chen et C. Guo, “Tiara : A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing,” dans *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA : USENIX Association, avr. 2022, p. 1345–1358. [En ligne]. Disponible : https://www.usenix.org/conference/nsdi22/presentation/zeng

[22] “Open Tofino,” Intel, 2020. [En ligne]. Disponible : https://github.com/barefootnetworks/Open-Tofino/blob/9b70ee58656bc32bdd144dbd1139b35ae6b0b5f2/PUBLIC_Tofino-Native-Arch.pdf

[23] C. Cascone, R. Bifulco, S. Pontarelli et A. Capone, “Relaxing state-access constraints in stateful programmable data planes,” *ACM SIGCOMM Computer Communication Review*, vol. 48, nº. 1, p. 3–9, 2018. [En ligne]. Disponible : https://doi.org/10.1145/3211852.3211854

[24] E. Schooler, G. Camarillo, M. Handley, J. Peterson, J. Rosenberg, A. Johnston, H. Schulzrinne et R. Sparks, “SIP : Session initiation protocol,” Internet Requests for Comments, IETF, RFC 3261, 2002, library Catalog : tools.ietf.org. [En ligne]. Disponible : https://tools.ietf.org/html/rfc3261

[25] “GPRS Tunnelling Protocol (GTP) across the Gn and Gp interface,” 3rd Generation Partnership Project, 3rd Generation Partnership Project, 3GPP TS29.060 V16.0.0, Mars 2020. [En ligne]. Disponible : https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1595

[26] R. Miao, H. Zeng, C. Kim, J. Lee et M. Yu, “SilkRoad : Making stateful layer-4 load balancing fast and cheap using switching ASICs,” dans *Proceedings of the Conference*

of the ACM Special Interest Group on Data Communication. ACM, 2017, p. 15–28. [En ligne]. Disponible : https://dl.acm.org/doi/10.1145/3098822.3098824

[27] The CAIDA UCSD , "Anonymized Internet Traces," 2019. [En ligne]. Disponible : https://www.caida.org/data/passive/passive_dataset.xml

[28] A. Arfeen, K. Pawlikowski, D. McNickle et A. Willig, "The role of the Weibull distribution in modelling traffic in Internet access and backbone core networks," *Journal of Network and Computer Applications*, vol. 141, p. 1–22, 2019. [En ligne]. Disponible : https://www.sciencedirect.com/science/article/pii/S1084804519301547