UNIVERSITÉ DU QUÉBEC À MONTRÉAL

ÉTUDE STRATÉGIQUE DE DÉPLOIEMENT DE COMPTEURS D'UN NŒUD DE COMMUNICATION BASÉE SUR DES APPROCHES DE TREILLIS

MÉMOIRE

PRÉSENTÉE

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

OMAR MOUNAOUAR

DÉCEMBRE 2013

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LEXIQUE

EXTENT	extension
EXTENT	Extension
FLOW	Flot de paquets
HEADER	En-tête de paquet
INTENT	Intension
LINE CARD	Carte de ligne
MATCH	Correspondance
MATCH FIELD	Champ de correspondance
METER ENTRY	Entrée de mesure
METER TABLE	Table des entrées de mesure

RÉSUMÉ

La surveillance de réseau, plus précisément l'observation du trafic (*traffic monitoring*), est une activité essentielle permettant aux opérateurs de bénéficier de la visibilité nécessaire aux opérations quotidiennes et à la planification requise de leur réseau, assurant ainsi l'atteinte d'une certaine qualité de service. Les solutions actuelles se relèvent très gourmandes en ressources du nœud de communication et ils n'exploitent pas le lien entre le traitement des paquets et les requêtes des utilisateurs.

Dans ce mémoire, nous proposons FLOWME, une nouvelle approche de surveillance des flots basée sur les treillis, qui améliore l'utilisation des ressources, en gardant un nombre minimum de compteurs matériels. FLOWME permet une interrogation fine des flots de trafic sur le réseau et une extraction exacte des résultats des requêtes demandées. Notre démarche consiste à découvrir les associations qui existent entre les flots traités par le nœud de communication et les requêtes des utilisateurs et à les exploiter pour construire un système de compteurs de taille minimale. L'idée de base est que ces associations nous permettent de trouver deux types de groupes de flots. Un premier type rassemble les flots qui participent à la réponse d'une même requête. Un deuxième type de groupes de flots est constitué des partitions de l'ensemble réponse des requêtes. L'affectation des compteurs à ces derniers groupes de flots nous permet de réduire l'effort de l'opération de mesure en le concentrant sur des groupes de flots disjoints au lieu des flots individuels. Les résultats de validation de FLOWME sur les distributions de trafic, de requêtes et d'entrées de flots de nos expériences montrent une réduction jusqu'à 90% sur le coût mémoire, et moins de $30ms$ en moyenne pour mettre à jour la structure de treillis par ajout d'un flot.

À l'issue de ce travail, nous avons montré que les treillis fournissent au domaine de la surveillance du trafic un moyen simple pour assurer une affectation optimale des compteurs matériels sur le nœud de communication et pour avoir la flexibilité requise pour s'adapter aux différents types d'applications et requêtes des administrateurs de réseaux.

INTRODUCTION

La surveillance de réseau, plus précisément l'observation du trafic (*traffic monitoring*), est une activité essentielle permettant aux opérateurs de bénéficier de la visibilité nécessaire aux opérations quotidiennes et à la planification requise de leur réseau, assurant ainsi l'atteinte d'une certaine qualité de service. Les outils d'observation de trafic requièrent une grande flexibilité afin de permettre la couverture de tous les types d'application et des requêtes d'observation des administrateurs [7]. Ces outils se révèlent très gourmands en terme de ressource (mémoires de recherche TCAM/DRAM, mémoire des paquets SRAM ainsi que du traitement de paquet) et engendrent des délais dans le traitement des paquets.

Étant donné la limite de ces ressources, les méthodes de surveillance usuellement utilisées sont basées sur une approche d'échantillonnage pour laquelle seulement une fraction du trafic est mesurée. L'approche la plus répandue est NetFlow [3] : elle exploite un échantillonnage progressif limité par les ressources disponibles. Toutefois, lors de montée en charge du trafic, les mesures obtenues perdent leur précision et donc leur utilité.

De plus, ces outils de surveillance de trafic n'ont pas la flexibilité requise pour s'adapter aux différents types d'application et aux requêtes des administrateurs de réseau. Des travaux récents [30; 13] cherchent à exploiter la structure des flots et la structure des requêtes des administrateurs pour élaborer une méthode de surveillance requérant un minimum de ressource. Un inconvénient majeur des solutions actuelles est qu'elles ignorent largement les associations qui existent entre la sémantique des requêtes et les descripteurs des flots traités par le nœud de communication. Nous considérons comme crucial ces associations et nous croyons que seule une structure qui les exprimera correctement aura la richesse et la flexibilité afin d'assurer une affectation optimale des compteurs sur le nœud de communication et minimisant ainsi les ressources requises pour cette surveillance. Un nœud de communication désigne tout équipement de réseau, ce qui peut être un routeur, un commutateur ou une passerelle.

La découverte des associations est une opération bien établie dans la littérature de *l'analyse formelle de concepts* (AFC) [11]. Le concept formel représente l'association entre des objets et leurs propriétés, la hiérarchie des concepts est appelée *treillis de concepts*. Les algorithmes

d'analyse des treillis de concepts sont conçus pour l'analyse et la découverte des associations dans un grand volume de données. La structure résultante de cette analyse contient toutes les associations découvertes. En plus, lorsque les données initiales évoluent, l'opération de mise à jour de la structure résultante est simplifiée et ne nécessite pas de reconstruction totale. Ces caractéristiques rendent les treillis de concepts particulièrement intéressants dans le cadre de la surveillance d'un grand nombre de flots de trafic.

Notre démarche consiste à découvrir les associations qui existent entre les flots traités par le nœud de communication et les requêtes, et ainsi les exploiter pour construire un système de compteurs de taille minimale. L'idée de base est que ces associations nous permettent de trouver deux types de groupes de flots. Un premier type rassemble les flots qui participent à la réponse d'une même requête. Ces ensembles peuvent se chevaucher si un flot participe dans la réponse de plusieurs requêtes. La classification dans le nœud de communication est un processus qui associe un paquet reçu à une entrée de flot unique. Pour exploiter ce mécanisme pour répondre aux requêtes, nous introduisons un deuxième type de groupes de flots constitué des partitions de l'ensemble réponse des requêtes. Le nombre des groupes de flots résultant est minimal. L'affectation des compteurs à ces groupes de flots nous permet de réduire l'effort de l'opération de mesure en le concentrant sur des groupes de flots disjoints au lieu des flots individuels.

Notre méthode de surveillance de trafic, basée sur les treillis, dite FLOWME, permet une interrogation fine des flots de trafic sur le réseau et une extraction exacte des résultats des requêtes demandées. Le système de compteurs construit est de taille minimale et permet à toutes les requêtes d'obtenir une réponse exacte. En outre, l'utilisation de la mémoire au niveau du nœud de communication est encore réduite, en se concentrant uniquement sur les flots utiles correspondant à des requêtes utilisateur.

Le présent mémoire est organisé comme suit:

Le chapitre 1 explore l'état de l'art de la surveillance des flots du trafic réseau et les défis que rencontrent les solutions actuelles, comme la demande en mémoire et en traitement de paquets.

Le chapitre 2 introduit les fondements théoriques de la stratégie choisie, basée sur les treillis, et définit formellement la tâche de découverte de l'affectation optimale des flots aux compteurs. Le nombre de compteurs établis de cette façon est minimal par rapport aux exigences

suivantes : répondre à toutes les requêtes actives, et chercher à ce qu'un flot participe tout au plus à un seul compteur matériel.

Dans le chapitre 3 nous proposons un ensemble d'algorithmes de construction/maintenance du treillis de concepts et sa sous-structure de projection. Les algorithmes de projection sont des méthodes originales qui réalisent la tâche centrale de partitionnement de l'ensemble total des flots en sous-ensembles disjoints. Un compteur matériel unique est ensuite associé à chacun de ces sous-ensembles.

Le chapitre 4 décrit la conception et l'implémentation de FLOWME. L'implémentation est constituée d'un collecteur et d'un nœud de communication pour les opérations de traitement de paquets et de surveillance des flots. Le collecteur est responsable de la construction et de la maintenance du treillis et du module d'affectation des compteurs en plus de la collecte des mesures et du calcul des valeurs des requêtes.

Le dernier chapitre décrit les résultats des expérimentations de notre stratégie de surveillance sur un nœud de communication physique. La réduction en nombre de compteurs par notre méthode surpasse largement l'affectation de compteur à chaque flot.

La conclusion fait le point sur les contributions de ce travail, les points forts et les points faibles de notre solution, et met en avant les perspectives envisagées.

CHAPITRE I

LA SURVEILLANCE DES FLOTS DU TRAFIC RÉSEAU

1.1 Introduction

La surveillance du trafic réseau est une activité essentielle, mais qui reste très gourmande en ressources du nœud de communication. Étant donné la complexité associée à cette tâche en termes de classification de paquets et de mise à jour des valeurs des compteurs, les méthodes de surveillance utilisées dans la pratique se basent sur une approche d'échantillonnage pour laquelle seulement une fraction du trafic est mesurée. Des travaux récents cherchent à exploiter la structure des flots et la structure des requêtes des administrateurs pour élaborer une méthode de surveillance requérant un minimum de ressource.

1.2 Le processus de surveillance des flots du trafic

La surveillance des flots du trafic est une tâche gourmande en ressources du nœud de communication. Dans cette section, nous explorons la complexité associée à cette tâche en termes de classification des paquets et mise à jour des compteurs.

1.2.1 La classification des paquet

Dans un nœud de communication, le processus de catégorisation des paquets en flots est appelé classification de paquets [15]. Un *flot* est l'ensemble des paquets qui sont décrits par des paramètres communs. L'ensemble de ces paramètres constitue le *descripteur de flot.* Le descripteur du flot peut être une entrée dans la TCAM. Dans OpenFlow [19], une *entrée de flot* est constituée du descripteur du flot et d'une suite d'actions, ce qui permet au nœud de communication de *traiter* les paquets appartenant au flot de la même manière.

La classification est l'une des tâches les plus coûteuses dans le nœud de communication. La classification est évaluée principalement par trois métriques : (1) la complexité de l'algorithme de classification dans le pire des cas, (2) le temps de mise à jour d'une nouvelle entrée de flot et (3) la capacité en mémoire demandée. La complexité de l'algorithme de la TCAM est de $O(1)$ ce qui correspond à un seul accès mémoire nécessaire pour classifier le paquet, pour l'algorithme 1-bit trie [25], la complexité est de $O(W)$ ce qui correspond à W accès mémoire dans le pire des cas, avec W la taille du descripteur du flot. Afin d'augmenter la bande passante, il faut réduire le temps total d'accès à la mémoire nécessaire pour traiter le paquet. Nous avons donc intérêt à adopter des stratégies de surveillance qui minimisent le recours à la classification de paquets. Le livre [2] donne plus de détails sur les différents algorithmes de classification.

1.2.2 La mise à jour des compteurs

La mise à jour des compteurs est une tâche très coûteuse en terme d'accès mémoire. Le nœud de communication ne peut pas mettre à jour les compteurs pour tous les paquets qui le traversent [7; 23]. Pour réaliser une mise à jour d'un compteur, le nœud de communication doit accéder à la mémoire une première fois pour lire la valeur courante du compteur, incrémenter cette valeur et accéder une deuxième fois à la mémoire pour écrire la nouvelle valeur du compteur. De cette manière, Devarat Shah et al. [23] estiment qu'il faut une mémoire de temps d'accès $P/(2CR)ns$ pour pouvoir mettre à jour les compteurs pour tous les paquets qui traversent un nœud réseau avec un débit d'arrivée R (en $Gbps$) et une taille minimale de paquets P (en bits), C est le nombre de compteurs incrémentés par chaque paquet. Par un calcul simple, nous déduisons que pour une DRAM de temps d'accès $84ns$, et pour un nœud de communication qui incrémente deux compteurs pour chaque paquet surveillé, le débit maximal est de $2,97Mpps$ ou l'équivalent de $1.52Gbps$ pour des paquets de 64 octets. Ce débit est très réduit par rapport aux débits des liens réseau. Les nœuds de communication actuels ne peuvent pas donc mettre à jour les compteurs pour tous les paquets.

1.3 L'échantillonnage du trafic

Le problème que nous avons relevé dans la section 1.2 est que le processus de surveillance des flots du trafic ne peut pas être exécuté pour chaque paquet reçu. La solution la plus adoptée NETFLOW, avancée par Cisco, adresse ce problème par une approche d'échantillonnage des paquets.

1.3.1 Le principe de l'échantillonnage

L'échantillonnage du trafic réseau est le processus de prélèvement partiel des paquets. La fréquence de prélèvement des paquets dans le nœud de communication est limitée par la capacité de ses ressources. Une grande fréquence d'échantillonnage nécessite plus d'accès à la mémoire et consomme donc la bande passante mémoire disponible pour les autres opérations de traitement de paquet. Si un échantillonnage n'est pas suffisamment précis, il peut conduire l'administrateur du réseau à des décisions erronées.

Les techniques d'échantillonnage utilisées dans les nœuds de communication actuelles sont l'échantillonnage statique et l'échantillonnage dynamique. l'échantillonnage statique utilise une fonction de probabilité fixée pour prélever un paquet parmi un ensemble de paquets de cardinalité fixe, le plus simple est de prélever le dernier paquet de cet ensemble. Cette technique est utilisée par les protocoles Cisco Sampled NetFlow et Juniper Packet Sampling. L'échantillonnage adaptatif, utilisé notamment par Adaptive NetFlow [5], prend en compte la variation du volume de données reçues pour contrôler efficacement la fréquence de prélèvement des paquets.

1.3.2 L'approche NetFlow

NETFLOW est un protocole qui permet le transfert des résultats de mesures par flot à partir des points de surveillance (Client NETFLOW) vers un point de surveillance central (Collecteur NETFLOW). NETFLOW, qui utilise une DRAM dans les routeurs de Cisco [3; 7], confronte deux problèmes, premièrement, l'accès à la mémoire centrale, qui est généralement commune à plusieurs fonctionnalités, ralentit le traitement des paquets. Deuxièmement, la bande passante est surchargée lorsque le nombre des flots à mesurer devient très grand, par exemple l'article [8] rapporte un taux de perte de 90% des messages de résultats de mesures NETFLOW en période de charge. La solution avancée par NETFLOW repose sur l'échantillonnage pour éviter de faire passer tout les paquets par la table de surveillance et réduire donc les besoins en calcul, accès mémoire et bande passante.

L'architecture de l'implémentation de NETFLOW sur le routeur Cisco Nexus 7000 est présentée dans la figure 1.1. Lorsque le premier paquet d'un flot est identifié par la table NETFLOW (NF Table), un enregistrement de flot est créé dans une mémoire cache locale à la line-card (Hardware NetFlow Creation). *L'enregistrement de flot* contient la description du flot ainsi que les résultats de mesures associées qui sont des compteurs sur le nombre des paquets du

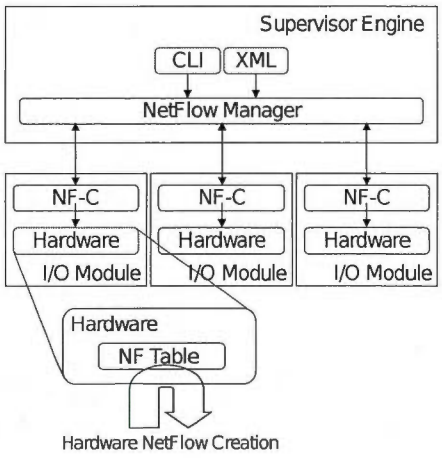


Figure 1.1: Architecture NETFLOW sur le routeur Cisco Nexus 7000 [24]

flot et le nombre d'octets total des paquets du flot. Les enregistrements de flots qui passent un certain temps dans la mémoire cache sont exportés par *le client local de surveillance* NETFLOW qui se trouve sur chaque *line-card* (NF-C). Le NetFlow Manager distribue la configuration sur les clients NETFLOW de chaque line-card, cette configuration est introduite soit par Interface de commande (CLI) ou par XML. Le Supervisor Engine centralise les résultats de mesures des différentes line-cards du routeur avant de les acheminer vers *le collecteur distant* NETFLOW qui est une base de données des mesures des enregistrements de flots. Les applications accèdent ensuite aux données du collecteur NETFLOW pour répondre aux requêtes des utilisateurs.

1.3.3 Les amélioration des performances du processus de surveillance

Un des problèmes majeurs que rencontre NETFLOW est l'explosion de la taille de la mémoire cache des statistiques. La mémoire cache est composée par des enregistrements de flots, chaque enregistrement est une structure de données qui garde des compteurs sur un flot surveillé. Un enregistrement de flot est créé dans la mémoire cache lorsque le nœud de communication reçoit le premier paquet du flot ensuite les autres paquets du flot mettent à jour les compteurs de l'enregistrement du flot. Le nombre des compteurs et donc la taille de la cache explose lorsque le nombre des flots augmente. Il faut donc chercher des stratégies pour réduire le nombre des compteurs.

Ghannadian *et al.* ont proposé dans leur brevet [13] de contrôler le nombre des enregistrements de flots créé en les filtrant avec une base de *measurement policies*. Définir ces filtres

est une tâche de l'administrateur du réseau ou d'une application externe. Par exemple, le filtre peut exprimer une bande passante minimale pour les flots qui seront mesurés. Dans ces solutions qui améliorent l'utilisation de la mémoire des compteurs, la précision des résultats, de la surveillance dépend toujours de la fréquence de l'échantillonnage.

FlexSample [21] propose de surveiller des sous-populations des flots qui sont définies *a priori*. Les sous-populations sont des groupes de flots définis par l'utilisateur. FlexSample ajuste dynamiquement la fréquence d'échantillonnage pour les paquets des groupes mesurés. FlexSample nécessite d'utiliser les filtres *Bloom* pour classifier les paquets, ce qui est d'après [22] une solution gourmande en mémoire. Cette méthode utilise en entrée des descripteurs statiques pour les groupes de flots. Adapter les descripteurs des groupes de flots en fonction des flots traités par le nœud de communication est une stratégie plus intéressante. Une telle stratégie va réduire le coût de classification des paquets pour la tâche de la surveillance des flots.

1.4 La surveillance des flots à l'extérieur du chemin de traitement des paquets

Afin d'éviter de surcharger le chemin du traitement des paquets du nœud de communication par les besoins en calcul et en accès mémoire des fonctionnalités de surveillance des flots, plusieurs solutions proposent de réaliser ces traitements à l'extérieur du chemin du traitement des paquets.

1.4.1 L'approche ProgME

ProgME [30] est une approche de surveillance qui cherche à exploiter en même temps la structure des flots et la structure des requêtes des administrateurs pour élaborer une méthode de surveillance requérant un minimum de ressources. Pour cette fin, ProgME propose premièrement un langage d'expression des requêtes des utilisateurs qui utilise les opérations ensemblistes $\cap$, $\cup$ et $\setminus$ pour composer des requêtes complexes à partir de requêtes simples. Deuxièmement, pour répondre à un ensemble de requêtes, ProgME les décompose en sous-ensembles disjoints de flots, appelés *flowsets*, et attribue ensuite un compteur à chacun d'eux.

ProgME n'exploite pas les associations entre les requêtes et les flots traités par le nœud de communication. Il génère ses propres entrées qui représentent les *flowsets* à partir des requêtes utilisateurs. Ces entrées générées ne serviront pas pour traiter les paquets, mais uniquement pour surveiller les flowsets. Les mêmes auteurs de l'article de ProgME ont proposé une im-

plémentation matérielle [18] qui utilise des blocs de traitement parallèle (Processing Elements) pour accélérer le processus de classification demandé par cette approche.

1.4.2 L'approche AUTOFOCUS

AUTOFOCUS [6] est un outil pour l'analyse hiérarchique hors-ligne du trafic. Il n'utilise pas des flots prédéfinis, mais il les découvre plutôt. À cette fin, il amorce une fouille d'associations dans les hiérarchies de valeurs généralisées fréquentes pour chaque champ d'en-tête dans la trace et les combine en une structure multidimensionnelle globale. La structure comprend à la fois les flots les plus importants en termes de volume de données et certains flots résiduels.

Cette fouille hors-ligne découvre les flots importants en analysant seulement les associations qui existent entre les descripteurs de paquets. Cette approche ne répond pas à des requêtes prédéfinies, mais découvre des motifs intéressants. Dans la pratique, nous avons connaissance des requêtes a priori. Le nœud de communication peut exploiter cette technique pour optimiser le placement des compteurs matériels. Le nœud de communication peut découvrir les groupes de flots intéressants pour la surveillance en utilisant une approche de fouille d'associations entre les flots traités par le nœud de communication et les requêtes.

1.5 La surveillance des flots au niveau du chemin de traitement des paquets

La surveillance des flots de paquets en utilisant les flots traités par le nœud de communication est un problème ouvert. Les solutions actuelles n'ont pas de stratégie qui exploite le lien entre le traitement des paquets et les requêtes des utilisateurs pour insérer les compteurs. Cela résulte en une double classification, puisqu'une partie de la classification nécessaire pour répondre aux requêtes pourrait être faite lors du traitement du paquet pour les opérations de filtrage, routage et commutation. Dans ces solutions, les associations requêtes-flots sont découvertes *a posteriori* au niveau du collecteur distant et des applications utilisateurs, il y a donc un volume important de données échangées sur des flots inutiles pour l'utilisateur.

Le système de surveillance doit limiter le nombre des compteurs à mettre à jour tout en étant capable de répondre à toutes les requêtes. Si le nombre des compteurs requis pour la surveillance est suffisamment petit, nous pouvons utiliser une mémoire rapide et faire passer tous les paquets par le processus de surveillance.

1.6 L'allocation des compteurs dans OpenFlow

Dans cette section, nous allons voir comment la surveillance des flots est implémentée par le protocole OpenFlow au niveau des tables des entrées de flots et au niveau des *meter entries*.

1.6.1 La table des flots et compteurs

Les tables de flots du protocole OpenFlow [19] sont composées par un ensemble d'entrées de flots. Chaque entrée de flot est composée de champs d'entêtes, de compteurs et d'une suite d'instructions. Les champs d'entêtes de l'entrée permettent d'identifier les paquets qui correspondent au même flot. Les compteurs mesurent le nombre de paquets et la taille totale du flot. Les instructions définissent les actions que le nœud de communication doit appliquer sur les paquets appartenant au flot. Les opérations sur les compteurs sont la lecture, l'incrémentation et l'initialisation.

La stratégie d'OpenFlow d'établissement des compteurs dans chaque entrée de flot est problématique, car il ne spécifie pas quel compteur activer. Dans une implémentation matérielle, il n'est pas possible d'activer tous les compteurs parce qu'à grands débits, le nœud de communication ne peut pas surveiller tous les flots de paquets. Il faut donc trouver une stratégie pour limiter le nombre des compteurs des entrées Openflow à mettre à jour.

La TCAM a une propriété intéressante : si un paquet peut correspondre à plusieurs entrées, elle sélectionne le seul résultat qui correspond a l'entrée possédant la plus grande priorité. L'article [17] propose une approche qui ajuste la priorité des entrées OpenFlow installées sur la TCAM pour maintenir des compteurs uniquement pour les flots de plus grande priorité.

1.6.2 Les compteurs de groupes d'entrées Openflow

OpenFlow dans sa version récente 1.3, propose de regrouper les compteurs des entrées de flots dans des enregistrements de flots appelés *meter entries*. Un *meter entry* surveille et contrôle le débit de toutes les entrées de flots qui lui sont rattachées. Lorsqu'une entrée de flot est associée à un *meter entry*, les paquets qui correspondent à l'entrée de flot vont incrémenter les compteurs du *meter entry* associé.

Cette stratégie de groupement de flots est intéressante parce qu'elle permet de maintenir des compteurs pour des groupes d'entrées de flots au lieu des entrées de flots individuelles.

OpenFlow ne propose pas de stratégie pour trouver ces groupes de flots. Une stratégie de regroupement de flots qui tient en compte les requêtes utilisateurs peut donner des groupes d'entrées de flots qui réduisent le nombre des compteurs.

1.7 Conclusion

La surveillance du trafic requiert des opérations de classification et mises à jour de compteurs qui sont gourmands en ressources. Les solutions de surveillance des flots se basent le plus souvent sur l'échantillonnage, d'autres approches récentes exploitent les requêtes utilisateurs pour réduire l'utilisation des ressources. Les solutions actuelles n'ont pas de stratégie qui exploite le lien entre le traitement des paquets et les requêtes pour insérer les compteurs. Dans la suite, nous allons exploiter les associations qui existent entre les requêtes utilisateurs et les flots de paquets traités par le nœud de communication pour placer les compteurs d'une façon optimale, et ceci sans rajouter de structures supplémentaires au niveau du processus de traitement des paquets.

CHAPITRE II

L'OPTIMISATION DE LA SURVEILLANCE DES FLOTS DU TRAFIC ET LA THÉORIE DES TREILLIS

2.1 Introduction

Dans ce chapitre, nous introduisons la démarche et les outils de résolution du problème d'optimisation de la surveillance des flots et nous passons en revue d'une façon condensée les fondements théoriques sur lesquels se base notre solution.

Dans le chapitre précédent, nous avons constaté que les solutions actuelles de surveillance de trafic ignorent largement les associations qui existent entre la sémantique des requêtes et les descripteurs des flots traités par le nœud de communication. Ici, nous utiliserons une structure de treillis qui exprime correctement ces associations et qui nous permettra par la suite une recherche d'une affectation optimale des compteurs.

2.2 Le problème d'optimisation du processus de surveillance des flots

Dans cette section, nous donnons les terminologies et notations nécessaires pour la formulation du problème d'optimisation du processus de surveillance des flots du trafic et pour l'introduction par la suite des fondements théoriques de la solution.

2.2.1 Les terminologies et les notations

En réseau, un flot est un ensemble de paquets ayant une source et une destination connues pendant une durée de temps déterminée. Mathématiquement, en tant que filtre, un flot est identifié par une entrée de flot f correspondante composée d'un ensemble de champs d'entêtes

$\{h_1, h_2, .., h_n\}$.

Soit $\mathcal{F}$ l'ensemble de tous les flots installés dans la table de flots d'un nœud de communication. Notons F un sous-ensemble arbitraire de flots de $\mathcal{F}$. Soit $\mathcal{H}$ l'ensemble de toutes les valeurs des champs d'entêtes des flots de $\mathcal{F}$ (en plus de quelques autres éventuellement). Maintenant, il y a une association naturelle entre $\mathcal{F}$ et $\mathcal{H}$: $f \in \mathcal{F}$ est associé à toute $h \in \mathcal{H}$ tel que h filtre un sur-ensemble des paquets que filtre f. Évidemment, f est associé à chacun de ses propres champs d'entêtes h_f. Pourtant, il pourrait y avoir plus d'entêtes h qui sont associés à une entrée de flot f sans faire formellement partie de sa définition. Par exemple, supposons un flot de la composition suivante:

$$f = \{[\text{Port d'entrée} = 1], [\text{IPv4 src} = 10/8], [\text{IPv4 dst} = 132.208.130.1]\}$$

Maintenant, f est associé non seulement avec les trois champs d'entêtes qu'elle possède, mais avec beaucoup d'autres, par exemple, $h = \{[\text{IPv4 dst} = 132.208.130/32]\}$. L'association ci-dessus peut être formalisée comme une relation d'incidence $\mathcal{M} \subseteq \mathcal{F} \times \mathcal{H}$. Le tableau 2.1 illustre un exemple exprimant la relation de huit flots et dix champs d'entêtes dont l'incidence est représentée comme un tableau de croix où les flots sont les rangées et les champs d'entêtes sont les colonnes.

2.2.2 La formulation du problème

L'optimisation de la surveillance des flots de trafic revient à garder l'utilisation des ressources utilisées par les mécanismes de surveillance à un minimum.

La première occupation des concepteurs de solutions de surveillance de trafic est de réduire le nombre des accès à la mémoire, en particulier, réduire le nombre des compteurs matériels auxquels les statistiques des flots sont reportées. La réduction du nombre de compteurs est faite en agrégeant le maximum de flots utiles dans un même sous-ensemble. Un sous-ensemble correspond à un compteur matériel unique. Pour éviter la duplication de l'information, et avoir un seul compteur au maximum par flot, les sous-ensembles de flots doivent être disjoints.

Supposons un ensemble Q de requêtes potentielles d'utilisateurs. Une requête utilisateur $q \in Q$ est un ensemble d'expressions régulières sur les champs d'entêtes des flots. Sans perte de généralité, nous pouvons supposer que nos requêtes sont composées de champs d'entêtes de l'ensemble $\mathcal{H}$. Pour le dire autrement, supposons que quelque soit Q, nous pouvons former un $\mathcal{H}$

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}
f_0	x			x			x		x	
f_1	x			x	x		x			x
f_2		x				x		x		
f_3		x				x		x		x
f_4	x						x		x	
f_5	x				x		x			x
f_6			x			x			x	x
f_7			x			x		x		x

h_1 - Port d'entrée = 1
h_2 - Port d'entrée = 2
h_3 - Port d'entrée = 3
h_4 - MAC src = MAC_1
h_5 - MAC dst = MAC_{12}
h_6 - IPv4 src = 132.208.130/32
h_7 - IPv4 src =10/8
h_8 - IPv4 dst = 10/8
h_9 - IPv4 dst = 132.208.130.1
h_{10} - Port dst couche 4 = 21

Tableau 2.1: Contexte d'entrée FHM

qui contient ses champs d'entêtes. Nous appellerons ces requêtes *compatibles* avec le contexte.

La formulation du problème d'optimisation de surveillance du trafic que nous introduisons consiste à, étant donné,

- un ensemble de flots $\mathcal{F}$ à surveiller
- un ensemble de requêtes utilisateurs Q

trouver,

- une partition de $\mathcal{F}$ de cardinalité minimale et qui est capable de répondre à toutes les requêtes des utilisateurs.

Dans la suite, nous proposons une approche basée sur la théorie des treillis permettant de donner une solution optimale au problème. L'approche explore le cadre théorique de *l'analyse formelle de concepts* [12] qui extrait des hiérarchies de groupes à partir de tables de données comme celui présenté dans le tableau 2.1. Pour commencer, une telle hiérarchie est d'abord construite sur l'ensemble de flots, ensuite les requêtes des utilisateurs sont mappées dans une sous-structure qui révèle enfin l'ensemble optimal de compteurs.

2.3 Les fondements théoriques

Dans cette section, nous passons en revue d'une façon condensée les fondements théoriques de l'analyse formelle des concepts. En utilisant ces fondements, nous construisons une hiérarchie de sous-ensembles de flots permettant par la suite de trouver une partition optimale des compteurs matériels. Nous introduisons les fondements théoriques en utilisant la terminologie et les notations que nous avons données dans la section 2.2.1.

2.3.1 Les contextes et les concepts des flots

Dans l'analyse formelle de concepts (AFC), un ensemble de données organisé en tableau croisé est appelé contexte formel. Un *contexte formel* est défini par un triplet $\mathcal{K} = (\mathcal{F}, \mathcal{H}, M)$ tel que $\mathcal{F}$ et $\mathcal{H}$ sont deux ensembles et M une relation entre $\mathcal{F}$ et $\mathcal{H}$. Le tableau 2.1 représente un contexte formel, les flots $f \in \mathcal{F}$ sont des objets et les champs d'entêtes $h \in \mathcal{H}$ sont leurs attributs. Comme indiqué auparavant, une paire $(f, h) \in \mathcal{M}$ est interprétée: *le flot f satisfait le champ d'entête h*. Étant donné un tel contexte, l'AFC dévoile la structure cachée de la relation d'incidence $\mathcal{M}$ en termes de *concepts* : ce sont les paires d'ensembles de flots et d'ensembles des champs d'entêtes qui sont en parfaite correspondance. Pour comprendre la signification de la correspondance entre les flots et les champs d'entêtes, nous avons besoin d'introduire d'abord une paire d'opérateurs pour formaliser l'incidence $\mathcal{M}$ entre les flots et les champs d'entêtes. Formellement, les opérateurs $'$ lient un ensemble de flots à l'ensemble de tous les champs d'entêtes communs et lient un ensemble de champs d'entêtes à l'ensemble des flots qui les satisfassent, respectivement, les opérateurs $'$ sont définis par :

$$' : \wp(\mathcal{F}) \rightarrow \wp(\mathcal{H}), F' = \{h \in \mathcal{H} \mid \forall f \in F, (f, h) \in M\}$$
$$' : \wp(\mathcal{H}) \rightarrow \wp(\mathcal{F}), H' = \{f \in \mathcal{F} \mid \forall h \in H, (f, h) \in M\}$$

Par exemple, $\{f_3, f_6\}' = \{h_6, h_{10}\}$ et $\{h_2, h_6\}' = \{f_2, f_3\}$.

Les opérateurs $'$ induisent une structure très particulière sur les deux ensembles des parties d'ensembles ($\wp(\mathcal{F})$ et $\wp(\mathcal{H})$) qui se manifeste dans les paires d'ensembles qui sont l'image l'un de l'autre par les opérateurs $'$. Ces paires sont appelées concepts formels. Un *concept formel* du contexte $(\mathcal{F}, \mathcal{H}, M)$ est défini par une paire (F, H) tel que pour tout $F \subseteq \mathcal{F}, H \subseteq \mathcal{H}, F' = H$ et $H' = F$. L'ensemble F est appelé *l'extension* et l'ensemble H est appelé *l'intension* du concept $c = (F, H)$. Un concept est donc défini par son extension et son intension, l'extension est composée de tous les objets appartenant au concept, alors que l'intension contient tous

les attributs partagés par ces objets. La définition du concept formel a le même sens que celui utilisé en logique. Dans le contexte exemple FHM (Flow Header Match) du tableau 2.1, $(\{f_3, f_6, f_7\}, \{h_6, h_{10}\})$ est un concept, alors que $(\{f_3, f_6\}, \{h_6, h_{10}\})$ ne l'est pas $(\{h_6, h_{10}\}' \neq \{f_3, f_6\})$. Désormais, $E(c)$ et $I(c)$ seront utilisées pour désigner l'extension et l'intension du concept c.

2.3.2 L'ordre partiel et le treillis des flots

Dans un contexte, un concept peut être plus général qu'un autre. Par exemple, dans notre FHM, le concept $(\{f_2, f_3, f_6, f_7\}, \{h_6\})$ représentant les flots de paquets IPv4 d'adresse 132.208.130/32 est plus général que le concept $(\{f_3, f_6, f_7\}, \{h_6, h_{10}\})$ représentant les paquets d'adresses IPv4 132.208.130/32 et ayant un port couche 4 destination 21. Le critère de généralité permet donc d'ordonner de façon partielle les concepts.

L'ensemble des concepts $\mathcal{C}_\mathcal{K}$ d'un contexte $\mathcal{K}$ est partiellement ordonné par l'inclusion des intensions/extensions: $(F_1, H_1) \leqslant_\mathcal{K} (F_2, H_2) \Leftrightarrow F_1 \subseteq F_2, H_2 \subseteq H_1$. Rappelons que pour être un ordre partiel, la relation $\leqslant_\mathcal{K}$ doit être réflexive, antisymétrique et transitive.

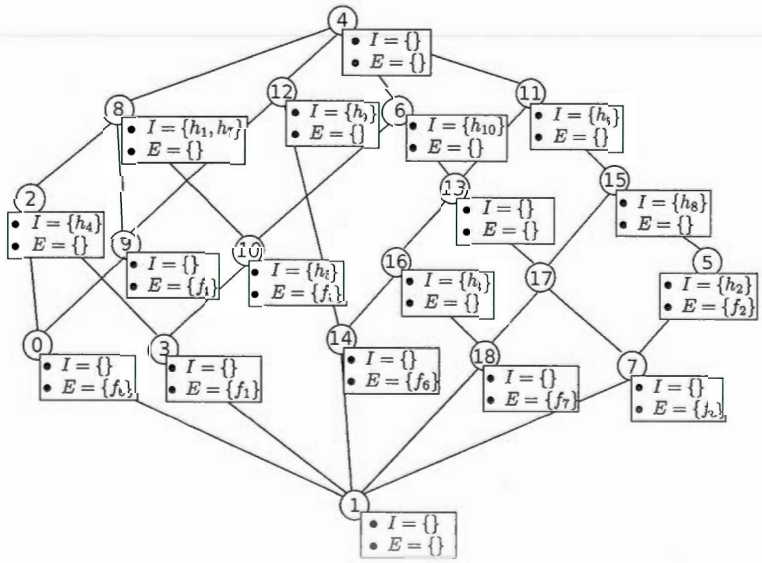


Figure 2.1: Treillis de concepts du contexte d'entrée FHM

La structure $\langle \mathcal{C}_\mathcal{K}, \leqslant_\mathcal{K} \rangle$ de notre FHM est représentée dans la figure 2.1 par son diagramme de Hasse, c'est-à-dire le graphe orienté acyclique induite par la relation de *précédence* $\prec_P$, qui

est la réduction transitive de $\leqslant_{\mathcal{K}}$. Désormais, nous allons identifier les concepts de notre FHM par leurs identifiants (voir la figure 2.1). Dans la figure 2.1, les étiquettes sont *réduites*, cela signifie que les informations qui peuvent être déduites ne sont pas représentées. La réduction des intensions/extensions de la représentation des concepts est faite pour augmenter la lisibilité (les objets sont hérités vers le haut et les attributs vers le bas). Ainsi, c_{13} apparemment vide est en fait $c_{13} = (\{f_3, f_6, f_7\}, \{h_6, h_{10}\})$. Le lecteur pourra se référer au tableau C.1 en appendice pour les étiquettes complètes. Dans la paire $c_2 \prec_{\mathcal{K}} c_8$, c_8 est appelé un *successeur immédiat* de c_2 et c_2 est un *prédécesseur immédiat* de c_8 (alors que c_0 est simplement un prédécesseur de c_8 dans $\langle \mathcal{C}_{\mathcal{K}}, \leqslant_{\mathcal{K}} \rangle$). Les ensembles de *prédécesseurs/successeurs immédiats* de c sont désignés c^L et c^U, respectivement. Par exemple, $c_8^L = \{c_2, c_9, c_{10}\}$. Les prédécesseurs communs/successeurs d'un ensemble A de concepts sont appelés minorants et majorants, respectivement.

Maintenant que nous avons vu à quoi les concepts ressemblent et comment ils sont hiérarchisés, nous arrivons à la principale structure de l'AFC, le *treillis de concepts* de $\mathcal{K}$. En effet, l'ensemble partiellement ordonné, dit *poset*, $\langle \mathcal{C}_{\mathcal{K}}, \leqslant_{\mathcal{K}} \rangle$ est un *treillis complet*, cela signifie que les minorants de tout ensemble ont un maximum et les supérieures un minimum. Formellement, dans le treillis de Galois $\mathcal{L}_{\mathcal{K}} = \langle \mathcal{C}_{\mathcal{K}}, \leqslant_{\mathcal{K}} \rangle$, pour tout $A \subseteq \mathcal{C}_{\mathcal{K}}$, le plus grand minorant $\bigwedge_{\mathcal{K}} A$ (dit *borne inférieure*) et le plus petit majorant $\bigvee_{\mathcal{K}} A$ (dit *borne supérieure*) existent dans le treillis. Par exemple, il peut être observé sur le schéma de la structure que $\bigvee_{\mathcal{K}} \{c_5, c_{14}, c_{18}\} = c_{11}$.

Les bornes inférieures sont identifiées par intersection sur les extensions des concepts alors que pour les bornes supérieures, la propriété de l'intersection est sur les intensions des concepts (voir le *Théorème fondamental* de l'AFC dans [11]). Dite autrement, une intersection des extensions (intensions) est également une extension (intension), ce qui induit une structure isomorphe au treillis de concepts sur la famille d'intensions, $\mathcal{C}_{\mathcal{K}}^h$, et celle des extensions, $\mathcal{C}_{\mathcal{K}}^f$. Les familles $\mathcal{C}_{\mathcal{K}}^h$ et $\mathcal{C}_{\mathcal{K}}^f$ sont donc fermées par intersections, donc les posets $\langle \mathcal{C}_{\mathcal{K}}^h, \subseteq \rangle$ et $\langle \mathcal{C}_{\mathcal{K}}^f, \subseteq \rangle$ sont des inf-demi-treillis. Nous rappelons qu'un demi-treillis est un ensemble ordonné avec seulement une des opérations du treillis ci-dessus (ici l'opération de borne inférieure). Le lecteur peut vérifier que l'extension de $c_7 = (\{f_3\}, \{h_2, h_6, h_8, h_{10}\})$ est l'intersection des extensions de $c_5 = (\{f_2, f_3\}, \{h_2, h_6, h_8\})$ et $c_{13} = (\{f_3, f_6, f_7\}, \{h_6, h_{10}\})$.

L'effet de la propriété ci-dessus ne peut être observé que si les extensions et les intensions de tous les concepts sont *étendues* à partir de ceux de la figure 2.1. En fait, la *réduction* d'étiquetage, détaillée précédemment, exploite une autre propriété intéressante du treillis de concepts énonçant que chaque $f \in \mathcal{F}$ ($h \in \mathcal{H}$) peut être attribué à un concept unique $c = (F, H)$

tel que $f \in F$ $(h \in H)$). Formellement, les fonctions μ et ν suivantes attribuent des concepts dédiés à chaque élément du contexte:

$$\mu : \mathcal{F} \rightarrow \mathcal{C}_{\mathcal{K}} \text{ avec } \mu(f) = (\{f\}'', \{f\}')$$
$$\nu : \mathcal{H} \rightarrow \mathcal{C}_{\mathcal{K}} \text{ avec } \nu(h) = (\{h\}', \{h\}'')$$

Ici $''$ représente une application dupliquée de l'opérateur $'$, c'est un opérateur qui envoie chacun des ensembles des parties $\wp(\mathcal{H})$ et $\wp(\mathcal{F})$ dans lui-même. Maintenant, $\mu(f)$ $(\nu(h))$ est le concept plus petit (grand) comprenant f (h) appelé le *concept flot* de f (concept champ d'entête de h). Par conséquent, chaque f (h) a besoin d'être mentionné seulement dans l'extension (intension) de $\mu(f)$ $(\nu(h))$ tandis que tous les autres concepts vont "l'hériter" de ce concept vers le haut (vers le bas).

En résumé, le treillis de concept factorise toutes les descriptions partagées entre un ensemble d'objets et un ensemble d'attributs en révélant *tous* les groupes d'objets (attributs) qui partagent le maximum d'attributs (objets).

2.4 La construction du treillis des flots

La construction du treillis des concepts est une tâche bien établie dans la littérature. Dans cette section, nous passons en revue une version de *NextNeighbor* dans [1] (p.35). L'algorithme de base de la construction du treillis de concepts initial demande en entrée seulement la description des flots donnée par le contexte d'entrée FHM et génère en sortie un treillis des concepts correspondant.

L'algorithme 1 construit le treillis de concepts à partir du sommet $(\mathcal{F}, \mathcal{F}')$ vers le bas $(\mathcal{H}', \mathcal{H})$, en générant les enfants du concept actuel $c = (F_c, H_c)$. À cette fin, il produit d'abord les extensions d'un plus grand ensemble de sous-concepts par l'intersection de F_c par les images de tous les champs d'entêtes *en dehors de* H_c. Il connecte ensuite comme enfants de c seulement les maximums de l'ensemble résultant (et les mets en file d'attente pour un traitement ultérieur). Par exemple, au concept $c_8 = (\{f_0, f_1, f_4, f_5\}, \{h_1, h_7\})$ dans la figure 2.1, les quatre extensions suivantes sont générées: $\varnothing$ (par intersections avec h'_2, h'_3, h'_6, et h'_8), $\{f_0, f_1\}$ (avec h'_4), $\{f_1, f_5\}$ (avec h'_5 et h'_{10}), et $\{f_0, f_4\}$ (avec h'_9). Les trois derniers sont maximaux, donc ils sont les extensions des concepts des enfants de c_8 (c_2, c_{10}, et c_9, respectivement).

L'algorithme 1 trouve son origine dans les résultats suivants (voir [27]): (1) pour un concept (F, H) avec son concept enfants (F_i, H_i) , les $H_i - H$, appelées *faces* du concept (par

exemple les faces de c_8 sont h_2 ,h_3 ,h_4 ,h_5 ,h_6 ,h_8 ,h_9 et h_{10}), sont deux à deux disjoints, et (2) $\forall h \in H_i - H$, $h' \cap F = F_i$. Par conséquent, l'algorithme construit tous les (F_i, H_i) à partir de (F, H) en produisant tout les $h' \cap F$ possibles pour les champs d'entêtes $h \in \mathcal{H} - H$ et en partitionnant ces h dans les faces de (F, H). Les champs d'entêtes non-faces de $\mathcal{H} - H$ génèrent de plus petites intersections qui sont supprimées.

Algorithm 1: Algorithme de construction du treillis

input : $\mathcal{K}(\mathcal{F}, \mathcal{H}, \mathcal{M})$

output: Ensemble des concepts liés C

1 $C \leftarrow \{(\mathcal{F}, \mathcal{F}')\}$;
2 **while** $C \neq \varnothing$ **do**
3 $c = (F_c, H_c) \leftarrow C.pop()$; // Choix d'un concept ayant un extent maximal // dans C.
4 $Children \leftarrow \varnothing$;
5 **foreach** h *in* $\mathcal{H} - H_c$ **do**
// Pour tout attribut n'étant pas dans l'extent.
6 $F_h = F_c \cap h'$; // Créer l'extent du concept combinant H_c et h.
7 **if** $\exists \bar{c} \in Children$ *s.t.* $E_{\bar{c}} = F_h$ **then**
8 $E_{\bar{c}} \leftarrow E_{\bar{c}} \cup \{h\}$; // Si un enfant à déjà cet extent, ajouter h // à son intent.
9 **else**
10 $Children \leftarrow Children \cup \{(F_h, H_c \cup \{h\})\}$; /* Sinon créer un nouveau enregistrement dans *Children* et l'utiliser avec F_h comme extent et $H_c \cup \{h\}$ intent. */
11 $c^L \leftarrow \max(Children)$; // Connecter c aux enfants maximaux par // en-dessous.
12 $C \leftarrow C \cup c^L$; // $\cup$ évite les doublons.

L'appendice A détaille l'exécution des trois premières itérations de l'algorithme.

2.5 L'actualisation du treillis des flots

Dans cette section, nous passons en revue l'actualisation du treillis lors de l'ajout d'un flot. L'algorithme 2 suit un schéma bien établi dans la littérature de l'AFC pour mettre à jour

le treillis des flots [1; 14] .

2.5.1 L'évolution du contexte d'entrée

Cette section décrit l'évolution du contexte d'entrée et du treillis des flots lors de l'ajout d'un nouveau flot.

Supposons que le FHM dans le tableau 2.1 est prolongé par un nouveau flot f_8 avec les champs d'entêtes h_2, h_7 et h_9 tels que décrit dans le tableau 2.2, et que l'ensemble de données résultant soit noté *eFHM*. En terme d'AFC, étendre le contexte $\mathcal{K} = (\mathcal{F}, \mathcal{H}, M)$ avec un nouveau

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}
f_0	x			x			x		x	
...	...									
f_7			x			x		x		x
f_8		x					x		x	

Tableau 2.2: Contexte étendu eFHM

flot f_n, revient à produire le contexte $\mathcal{K}_n = (\mathcal{F}_n, \mathcal{H}_n, \mathcal{M}_n)$ tel que $\mathcal{F}_n = \mathcal{F} \cup \{f_n\}$, $\mathcal{H}_n = \mathcal{H}$ et $\mathcal{M} = \mathcal{M}_n \cap \mathcal{F} \times \mathcal{H}$.

Maintenant, le treillis des concepts du contexte eFHM nouvellement crée est présenté sur la figure 2.2. Pour faciliter l'illustration, nous avons fourni des étiquettes étalées des concepts nouvellement créés ou qui ont vu leurs extensions/intensions changer lors de l'actualisation. Le lecteur pourra se référer au tableau C.2 en appendice pour les étiquettes de tous les concepts du treillis.

Il est évident que le nouveau treillis peut être obtenu par construction à partir du contexte eFHM. Une comparaison rapide des deux figures 2.1 et 2.2 révèle qu'il y a beaucoup de structures en commun, en effet, dans les deux diagrammes, nous avons utilisé des numéros identiques pour des concepts qui sont homologues (Les concepts qui sont les mêmes dans les deux figures sont affichés sans préciser leurs intensions/extensions).

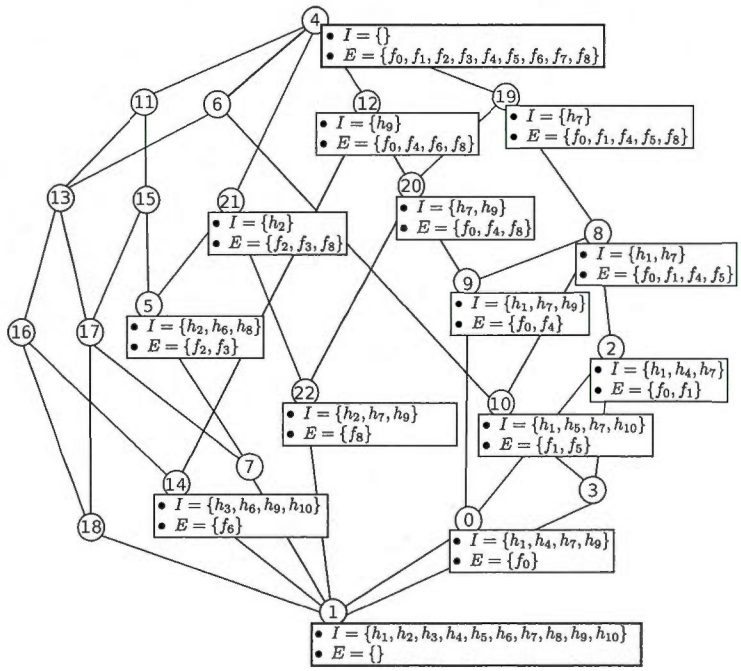


Figure 2.2: Treillis de concepts eFHM étendu par ajout du flot f_8.

2.5.2 L'évolution du treillis de flots

L'objectif de cette section est de décrire l'obtention du nouveau treillis de flots sans recommencer le calcul de tous ces éléments.

La caractérisation formelle des concepts du treillis des flots actualisé est présentée dans l'appendice B.2. L'idée de base de l'actualisation des concepts du treillis initial revient à utiliser l'intersection, notée H, de f'_n avec les intensions de $\mathcal{C}^h_{\mathcal{K}}$ pour modifier des concepts qui ne respectent plus la définition formelle d'un concept ou de générer de nouveaux concepts si besoin est. Une intersection H pourrai être déjà une intension dans l'ancien treillis, c'est-à-dire, soit un élément de $\mathcal{C}^h_{\mathcal{K}}$, ou non. Dans le cas où l'intersections H se trouve en dehors de l'ensemble $\mathcal{C}^h_{\mathcal{K}}$, l'algorithme doit générer un *nouveau concept*. Dans notre exemple, les intersections de $f'_8 = \{h_2, h_7, h_9\}$ avec $\mathcal{C}^h_{\mathcal{K}}$, qui ne sont pas intensions, sont $\{h_7\}$, $\{h_7, h_9\}$, $\{h_2\}$, et $\{h_2, h_7, h_9\}$. Ces intersections génèrent les concepts c_{19} à c_{22} spécifiques à $\mathcal{K}_n$. Remarquons que c_{22} est le concept-flot de f_8 dans $\mathcal{K}_n$.

Dans le cas où H existe déjà dans $\mathcal{C}^h_{\mathcal{K}}$, le concept (H', H) de $\mathcal{C}_{\mathcal{K}}$ doit être *modifié* en

ajoutant f_n à son extension pour qu'il satisfasse la définition de concept (voir section 2.3.1) dans $C_{\mathcal{K}_n}$. Une intersection H peut être générée par plusieurs concepts du treillis de départ, par exemple $\{f_8'\} \cap I(c_0) = \{h_7, h_9\}$ et $\{f_8'\} \cap I(c_9) = \{h_7, h_9\}$ (rappelons que $I(c_0) = \{h_1, h_4, h_7, h_9\}$ et $I(c_9) = \{h_1, h_7, h_9\}$). La propriété B.12 (ligne 2) nous indique quel concept parmi eux sera modifié par ajout du nouveau flot à son extension, il s'agit du concept c_m dont l'intension est la fermeture de H dans $\mathcal{K}$. Dans l'exemple, c'est le concept c_9 qui sera modifié car dans $\mathcal{K}$, $H'' = \{h_7, h_9\}'' = \{h_1, h_7, h_9\} = I(c_9)$. Notez que dans le treillis initial, le concept modifié $c_m = (F_m, H_m)$ est le concept maximal qui satisfait la relation $H = H_m \cap \{f_n\}'$ puisque son intension H_m est minimale parmi les intensions des concepts candidats.

2.5.3 L'algorithme d'actualisation du treillis des flots

L'algorithme d'actualisation du treillis des flots se décompose en trois blocs: calcul des intersections H (ligne 3), extension des concepts modifiées par ajout de f_n (lignes 4-5) et création de nouveaux concepts (lignes 6-10). Le tableau 2.3 représente la trace de l'exécution de l'algorithme 3 après traitement de chaque concept c de la liste ordonnée C par les instructions de la boucle *for* (ligne 2).

L'algorithme 2 commence par la liste des concepts (C) ordonnée par *taille croissante des intensions*, ce qui assure, étant donné une intersection H, de trouver en premier le concept maximal qui la génère.

Dans une boucle globale sur les concepts, l'algorithme produit toutes les intersections H possibles (ligne 3). Si H est déjà une intension dans $\mathcal{K}$ (ligne 4), le test est simplifié grâce à l'ordre dans la liste des concepts C, en effet, l'inclusion de l'intension du concept courant est suffisante. Si c'est le cas, le concept est *modifié* en ajoutant f_n à son extension (ligne 5). Par ailleurs, l'ensemble M des concepts modifiés est directement mis à jour pour permettre des tests rapides des nouvelles intersections H.

Le test de la ligne 4 décide si le concept courant doit être modifié ou non. Pour le premier concept c_4, $I_{c_4} = \emptyset$ et $f_n' = \{h_2, h_7, h_9\}$ donc $H = \emptyset$. $H = I_{c_4}$, donc c_4 est modifiée par ajout de f_8 à son extension. Pour c_6, $I_{c_6} = h_{10}$ et $H = \emptyset$, donc c_6 ne peut pas être modifiée et ne va pas donner naissance à un nouveau concept. Les concepts qui repondent au test de la ligne 4 sont c_4 et c_{12}.

c	I_c	H	E_c	c_n	E_{c_n}
c_4			$f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8$		
c_6	h_{10}		f_1, f_3, f_5, f_6, f_7		
c_8	h_1, h_7	h_7	f_0, f_1, f_4, f_5	c_{19}	f_0, f_1, f_4, f_5, f_8
c_{11}	h_6		f_2, f_3, f_6, f_7		
c_{12}	h_9	h_9	f_0, f_4, f_6, f_8		
c_{13}	h_6, h_{10}		f_3, f_6, f_7		
c_{15}	h_6, h_8		f_2, f_3, f_7		
c_2	h_1, h_4, h_7	h_7	f_0, f_1		
c_9	h_1, h_7, h_9	h_7, h_9	f_0, f_4	c_{20}	f_0, f_4, f_8
c_5	h_2, h_6, h_8	h_2	f_2, f_3	c_{21}	f_2, f_3, f_8
c_{10}	h_1, h_5, h_7, h_{10}	h_7	f_1, f_5		
c_{16}	h_3, h_6, h_{10}		f_6, f_7		
c_{17}	h_6, h_8, h_{10}		f_3, f_7		
c_0	h_1, h_4, h_7, h_9	h_7, h_9	f_0		
c_3	$h_1, h_4, h_5, h_7, h_{10}$	h_7	f_1		
c_7	h_2, h_6, h_8, h_{10}	h_2	f_3		
c_{14}	h_3, h_6, h_9, h_{10}		f_6		
c_{18}	h_3, h_6, h_8, h_{10}		f_7		
c_1	$h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, h_{10}$	h_2, h_7, h_9		c_{22}	f_8

Tableau 2.3: Trace de l'execution de l'algorithme 2

Pour savoir si H existe dans C, il suffit de regarder dans M, en effet, si H est intent dans C, on le saura dès la première génération de H par un concept c. Or, grâce a l'ordre croissant des intents, c'est toujours l'intent minimal pour H qui est rencontré en premier. Pour un H existant dans C c'est le concept modifié qui sera premier dans la liste. Ainsi, dans le deuxième cas (ligne 6), H n'est pas une intension ni dans les concepts de l'ancien treillis ni dans les concepts nouvellement générés (ensembles C_n et M). Une telle intersection déclenche la création d'un nouveau concept c_n (lignes 6 à 9). L'intension de c_n est exactement H, alors que son extension comprend l'extension du concept qui a généré cette intension (dit le *géniteur*) en plus de f_n (ligne 7).

Algorithm 2: Mise à jour du treillis par ajout de flot

input : Entrée de flot f_n, Listes des concepts C

output : Listes des concepts C

1 $C_n \leftarrow \varnothing$; $M \leftarrow \varnothing$; // Initialisation

2 **foreach** $c = (E_c, I_c)$ *in* C **do**

// Parcours descendant par taille des intents décroissante.

3 $H \leftarrow I_c \cap f'_n$; // Production des intersections.

4 **if** $H = I_c$ **then**

// test d'inclusion d'intent dans la description du nouveau

// flot.

5 $c \leftarrow (E_c \cup \{f_n\}, I_c)$; $M \leftarrow M \cup \{c\}$; // Modification du concept.

6 **else if** $\nexists \tilde{c} \in C_n \cup M$ *s.t.* $I_{\tilde{c}} = H$ **then**

// Teste si l'intersection est produite pour la première fois.

7 $C_n \leftarrow C_n \cup \{c_n = (E_c \cup \{f_n\}, H)\}$; // Création d'un nouveau concept.

8 $c^U \leftarrow c^U \cup \{c_n\}$; $c_n^L \leftarrow c_n^L \cup \{c\}$; // Connexion du géniteur et du

// nouveau.

9 $UpdateOrder(c_n, c^U, C_n, M)$; // Connexion du nouveau et

// successeurs immédiats.

10 $N \leftarrow (c, c_n)$

11 $C \leftarrow C \cup C_n$;

Le test de la ligne 6 décide si le concept est un géniteur. Si oui, alors un nouveau concept doit être créé. Pour c_8, $H = \{h_7\}$. c_8 ne sera donc pas modifiée car $H \neq I_{c_8}$. Par contre, c_8 est le premier concept tel que $H = \{h_7\}$ (test ligne 6), donc il est géniteur. Ainsi, c_8 donne naissance au concept c_{19} d'intension H et d'extension $E(c_8) \cup \{f_8\}$. Dans notre exemple, c_{19}, c_{20}, c_{21}, et c_{22} sont de nouveaux concepts avec les géniteurs c_8, c_9, c_5, et c_1, respectivement. Parmi eux, c_{22} est le concept-flot de f_8.

Pour mettre à jour les liens de précédence $\prec_{\mathcal{K}}$ avec la primitive *UpdateOrder*, d'abord chaque nouveau concept c_n est connecté au dessous de son géniteur (ligne 8). Ensuite, pour chaque parent $\tilde{c}$ de c, on cherche le concept dans $C_n \bigcup M$ qui a comme intent $I(\tilde{c}) \bigcap f'_n$ et parmi la liste resultante, on prend comme c_n^U les concepts minimaux. De plus, il faut vérifier si chaque $\tilde{c}$ est parent de c, le géniteur de c_n. Si c'est le cas, supprimer le lien entre $\tilde{c}$ et c.

Dans la ligne 10, nous gardons l'association entre les nouveaux concepts et leur géniteur parce qu'elle nous servira par la suite dans notre démarche (voir algorithme 4). Enfin, dans la ligne 11, nous complétons l'ensemble des concepts C par les nouveaux concepts C_n.

Pour résumer, étendre $\mathcal{C}_{\mathcal{K}}$ à $\mathcal{C}_{\mathcal{K}_n}$ est possible et permet d'économiser un grand effort de reconstruction du treillis du départ.

2.6 Conclusion

Dans ce chapitre, nous avons formulé le problème d'optimisation de la surveillance des flots du trafic et nous avons introduit la construction du treillis des flots, qui est une hiérarchie de sous-ensembles qui nous permettra par la suite de trouver la partition optimale des compteurs matériels. Dans la suite, nous allons donner un ensemble d'algorithmes permettant d'identifier la partition optimale de sous ensembles de flots à superviser et d'affecter des compteurs et calculer les valeurs des requêtes utilisateurs.

CHAPITRE III

L'AFFECTATION DES COMPTEURS

3.1 Introduction

Nous proposons dans ce chapitre un ensemble d'algorithmes d'identification et de mise à jour de la partition optimale des flots à superviser par les compteurs. Cette partition correspond à une sous-structure du treillis des flots introduite dans le chapitre précédent. Les propriétés des éléments de cette sous-structure permettent de : simplifier les algorithmes, sa construction et sa mise à jour, et donc d'assurer que l'évolution du système par ajout/suppression de flot ne demande pas une reconstruction totale.

3.2 Le demi-treillis des projections

Dans cette section, nous introduisons le demi-treillis des projections qui contient les sous-ensembles de flots à superviser.

Compte tenu d'un contexte $\mathcal{K} = (\mathcal{F}, \mathcal{H}, \mathcal{M})$ et un ensemble de requêtes $Q \subseteq \wp(\mathcal{H})$ ($\mathcal{H}$ est étendue au besoin pour assurer que le treillis contient tous les champs d'entêtes dans les requêtes), notre approche identifie essentiellement trois types de concepts de $\mathcal{C}_{\mathcal{K}}$: (1) concepts cibles, (2) concepts projections et (3) concepts bases. Tout d'abord, chaque $q \in Q$ est associée au concept dont l'extension est l'ensemble des flots satisfaisant q. Remarquant que $q \subseteq \mathcal{H}$, on peut donc écrire $\{q\}'$. Le concept $(\{q\}', \{q\}'')$ est appelé *concept cible* de q. Dans la suite, l'ensemble des concepts cibles sera noté $T(Q)$.

Nous avons besoin par la suite de partitionner les ensembles de flots réponses, c'est-à-dire, de partager toutes les extensions des concepts cibles en factorisant les parties communes, nous considérons donc les bornes inférieures des concepts dans $T(Q)$. On obtient par la suite un

plus grand ensemble de concepts: *les concepts projection* $P(Q)$ qui sont les bornes inférieures des concepts cibles. $P(Q)$ définit toutes les partitions admissibles des extensions des concepts cibles. Nous cherchons dans notre stratégie à trouver la partition optimale de l'ensemble des partitions $P(Q)$. La partition optimale est identifiée par un autre sous-ensemble de $P(Q)$ appelé *les concepts projections bases* notés $G(Q)$ qui sont reliés aux flots dans $\mathcal{F}$. Nous proposons une formulation mathématique de ces ensembles de concepts.

3.3 La caractérisation du demi-treillis des projections

Après avoir vu comment on génère le treillis des flots, nous allons maintenant identifier la sous-structure du treillis qui nous indiquera l'affectation optimale des compteurs. L'objectif est donc d'identifier T, P et G définis dans la section 3.2, à partir de l'ensemble des concepts $\mathcal{C}_\mathcal{K}$.

3.3.1 L'ensemble des flots de réponse

L'ensemble des flots de réponses d'une requête q est l'extension d'un concept dans $\mathcal{C}_\mathcal{K}$. Nous appelons ce concept, dont l'extension est l'ensemble des flots satisfaisant q, un *concept cible*, et on le note $\gamma(q)$. Inversement, on note $t(c)$ l'ensemble des requêtes satisfaites par l'extension du concept c. D'apres la définition B.1, le concept cible de q est $c \in \mathcal{C}_\mathcal{K}$ tel que $I(c) = q''$, ou de façon équivalente $E(c) = q'$. L'opération d'identification des concepts cibles est donc simple, elle consiste à trouver pour chaque requête q le concept c maximal pour lequel $q \subseteq I(c)$. Par exemple, avec Q dans le tableau 3.1 et $\mathcal{L}_\mathcal{K}$ dans la figure 2.1, $\gamma(q_2) = c_5$, et $\gamma(q_3) = \gamma(q_5) = c_8$ (les ensembles réponse $\{f_2, f_3\}$ et $\{f_0, f_1, f_4, f_5\}$, respectivement). L'ensemble des concepts cibles est donc $T(Q) = \{c_2, c_5, c_6, c_8\}$.

3.3.2 La partition optimale en sous ensembles de flots

L'ensemble des concepts cibles $T(Q)$ ne définit pas lui-même l'affectation des compteurs. Une solution est de mettre un compteur par $c \in T(Q)$, ce qui rendra les statistiques directement disponibles (sans aucune autre manipulation de compteurs). Cette solution violerait la contrainte qu'un flot est supervisé par au plus un compteur, en particulier lorsqu'un flot se trouve dans les extensions de plusieurs concepts cibles (par exemple, f_5 est à la fois dans c_6 et c_8). Dans l'autre extrême, affecter un compteur par flot n'est pas optimal, il est donc clair qu'il faut un compromis avec des compteurs affectés à des ensembles de flots qui, en terme d'inclusion

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9	h_{10}
f_0	x			x			x		x	
...	...									
f_6			x			x			x	x
f_7			x			x		x		x
q_1										x
q_2		x				x		x		
q_3	x									
q_4	x			x			x			
q_5							x			

h_1 - Port d'entrée = 1 h_6 - IPv4 src = 132.208.130/32

h_2 - Port d'entrée = 2 h_7 - IPv4 src =10/8

h_3 - Port d'entrée = 3 h_8 - IPv4 dst = 10/8

h_4 - MAC src = MAC_1 h_9 - IPv4 dst = 132.208.130.1

h_5 - MAC dst = MAC_{12} h_{10} - Layer 4 dst port = 21

Tableau 3.1: Le FHM et l'ensemble des requêtes Q de notre exemple de départ

ensembliste, se trouvent entre les ensembles réponse des requêtes et les flots individuels.

Notre approche se concentre sur les flots partagés entre les extensions des concepts cibles. L'idée est de diviser chaque extension en sous-ensembles et d'attribuer un compteur à chacun. Évidemment, certains sous-ensembles risquent de se retrouver dans plusieurs partitions de requêtes. Une telle répétition est bénéfique, car elle permet de "réutiliser" le même compteur pour plusieurs requêtes. Notre espoir est donc de trouver une partition globale de $\mathcal{F}$ qui maximise les "répétitions", ce qui revient à minimiser sa taille. Avec cette solution, le calcul de statistiques sur les requêtes consiste à sommer les compteurs. En outre, chaque extension dans $T(Q)$ est *partitionnée* pour éviter les chevauchements et des calculs arithmétiques complexes. En revanche, il ne devrait y avoir aucune redondance de flots entre les partitions locales. La partition en sous ensemble de flots correcte de $\mathcal{F}$ compose des partitions locales exactes pour chaque concept cible. Pour être optimale, la partition doit être de cardinalité minimale.

Pour trouver cette partition, il nous faut factoriser les intersections *arbitraires* des extensions des concepts cibles. Une propriété pratique du treillis de concepts entre en jeu ici: en raison de la fermeture de $\mathcal{C}^f_{\mathcal{K}}$, toutes les intersections sont eux-mêmes intensions des concepts

dans $\mathcal{L}_\mathcal{K}$. Nous appelons les concepts qui factorisent les parties communes des extensions des concepts cibles, des *concepts projection*. Les concepts projection donnent toutes les partitions des extensions des concepts cibles.

Afin de simplifier l'identification des concepts projection, nous introduisons la notion de vecteur requête $v(c)$ d'un concept c (voir la définition B.4). $v(c)$ indique les requêtes satisfaites par $I(c)$, par exemple pour c si $q_i \subseteq I(c)$ alors $v(c)[i] = 1$. $v(c)$ exprime les requêtes q dont les concepts cibles $\gamma(q)$ sont supérieur a c. Par exemple pour c_{10}, $v(c_{10}) = 10101$ exprime les requêtes q_1, q_3 et q_5 que leurs concepts cibles c_6 et c_8 sont les concepts cibles supérieurs a c_{10} ($\gamma(q_1) = c_6$,$\gamma(q_3) = c_8$ et $\gamma(q_5) = c_8$).

Vérifier qu'un concept c est la borne inférieure des concepts cibles qui lui sont supérieurs, revient à vérifier si son vecteur requêtes contient des 1 dans un ensemble de positions qui inclut strictement les positions à 1 dans chacun des vecteurs de ses successeurs immédiats (voir propriété B.5). Noter que si $q \subseteq I_c$ et $\bar{c} \leqslant c$, alors $q \subseteq I(\bar{c})$. En terme de vecteurs $v(c)[i] = 1 \Rightarrow v(\bar{c})[i] = 1$ pour tout i et tout c, $\bar{c}$ tel que $\bar{c} \leqslant c$. Formellement, $|v(c)| > \max_{\bar{c} \in c^U} (|v(\bar{c})|)$, avec c^U l'ensemble des parents de c et $v(\bar{c})$ le vecteur de bits indiquant les requêtes satisfaites par le concept $\bar{c}$. Cette inégalité dis que c est maximal parmi son voisinage pour l'ensemble des requêtes q dont les $\gamma(q)$ sont au-dessus de c. En d'autres termes, c est la $\bigwedge$ de ses concepts cibles. La propriété interessante ici est que $|v(c)| > \max_{\bar{c} \in c^U} (|v(\bar{c})|)$ ssi $c = \bigwedge \gamma(q_i)$ tel que $v(c)[i] = 1$. Dans notre exemple, $P(Q) = \{c_1, c_2, c_3, c_5, c_6, c_7, c_8, c_{10}\}$.

Maintenant, nous avons déterminé toutes les partitions des extensions des concepts cibles. Une propriété intéressante est qu'il existe un concept minimal unique qui détient un flot donné f, nous appelons ce concept le *concept base* du flot f, noté $\mu_Q(f)$. Notez que le concept base d'un flot peut être concept base pour plusieurs autres flots en même temps.

En utilisant la caractérisation formelle développée dans l'appendice B.1.3 du concept base : le concept base de f est la borne inférieure des projections qui détiennent un flot f. Inversement, l'ensembe des flots pour lesquels c est le concept base est noté $g(c)$. Un concept c est donc un concept base s'il est base pour au moins un flot f de son extension. Pour un tel f, nous avons $c = \mu_Q(f)$. À noter que si aucune requête ne porte sur f, le flot est associé au sommet du treillis T.

Du point de vu algorithmique, il est possible de tester si une projection c est base en la comparant avec les concepts flots des flots de son extension $E(c)$. Une façon rapide est de com-

parer les vecteurs requêtes $v(c)$ avec les $v(\mu_Q(f))$ de son extensions et de vérifier que les deux sont identiques. Le tableau 3.2 donne μ_Q pour notre exemple avec $G(Q) = \{c_2, c_3, c_5, c_6, c_7, c_8, c_{10}\}$.

Flot	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7
Base	c_2	c_3	c_5	c_7	c_8	c_{10}	c_6	c_6

Tableau 3.2: Correspondance entre flots et concepts-bases

La partition optimale pour l'exemple de base se compose des ensembles $\{f_0\}$, $\{f_1\}$, $\{f_2\}$, $\{f_3\}$, $\{f_4\}$, $\{f_5\}$ et $\{f_6, f_7\}$.

3.4 L'identification du demi-treillis des projections

Dans cette section, nous introduisons l'algorithme d'identification du demi-treillis des projections. Nous utilisons les caractéristiques des concepts cibles, projections et bases introduites dans la section 3.3. Notons $t(c)$ l'ensemble des requêtes dont le concept cible est c et $g(c)$ l'ensemble des flots dont le concept base est c.

L'algorithme 3 identifie l'appartenance des concepts aux ensembles P, T et G le long d'un parcours de haut en bas du treillis. À partir du haut du treillis (rappelons que l'extension du concept top contient tous les flots, et son intension contient les champs d'entêtes partagés par tous les flots) l'algorithme commence par identifier les concepts-cibles $\gamma(q)$, et propage leurs q dans les concepts enfants pour permettre le calcul progressif de $v(c)$ pour chaque c. Ensuite, l'algorithme vérifie l'appartenance des concepts à P et G.

L'algorithme 3 se décompose en quatre blocs: tri de la liste des concepts (ligne 1), identification des concepts cibles (lignes 3-6), identification des concepts projections (lignes 7-9) et identification des concepts bases (lignes 10-13). Le tableau 3.3 représente la trace de l'exécution de l'algorithme 3 après traitement de chaque concept c de la liste ordonnée C par les instructions de la boucle *for* (ligne 2).

Tout d'abord, la liste des concepts C est triée dans l'ordre des tailles des extensions (ligne 1) cela a pour objectif d'assurer que le premier concept dont l'intension correspond à $q \in Q$ est exactement le concept-cible de cette requête: $\gamma(q)$ (voir la définition B.1). Rappelons que $\gamma(q)$ est le concept cible de q s'il est le concept d'intension minimale qui inclut q.

Algorithm 3: Algorithme d'identification des partitions de flots

input : Liste des concepts C,

Ensemble de requêtes $Q = (q_1, .., q_i, .., q_n)$

output: Ensembles cibles, projections et bases (T, P, G)

1 $Sort(C)$;
2 **foreach** c *in* C **do**
3 **for** q_i *in* Q **do**
4 **if** $q_i \subseteq I(c)$ **then**
5 $v(c)[i] \leftarrow 1$;
6 $T \leftarrow T \cup \{c\}$; $Q \leftarrow Q - \{q_i\}$;
7 $v(c) \leftarrow v(c) \cup \bigcup_{\overline{c} \in c^U} v(\overline{c})$;
8 **if** $|v(c)| > \max_{\overline{c} \in c^U} (|v(\overline{c})|)$ **then**
9 $P \leftarrow P \cup \{c\}$;
10 **if** $|E^r(c)|=1$ **then**
11 **for** $p \in P$ **do**
12 **if** $v(p) = v(c)$ **then**
13 $G \leftarrow G \cup \{p\}$; $break()$;

Le test de la ligne 4 décide si le concept c est le concept cible de q_i. Pour le premier concept de la liste, c_4, aucune requête ne satisfait son intension donc $c_4 \notin T$. Par contre, le concept c_6 satisfait q_1, parce que $q_1 \subseteq Ic_6$ ($q_1 = Ic_6 = \{h_{10}\}$). c_6 est concept base pour q_1, donc $c_6 \in T$ et $t(c_6) = \{q_1\}$. La requête q_1 est par la suite retirée de Q et le bit $v(c_6)[1]$ est mis à 1 pour refléter le fait que c_6 répond à q_1 (ligne 6). De la même façon, c_6, c_8, c_2 et c_5 sont ajoutés à T parceque'ils sont les cibles des ensembles de requêtes $\{q_1\}$, $\{q_3, q_5\}$, $\{q_4\}$ et $\{q_2\}$, respectivement.

Un concept enfant satisfait tout les requêtes que ses parents satisfont. La ligne 7 reflète ce fait en fusionnant la partie locale (le concept est éventuellement cible pour certaines requêtes, ligne 5) avec les valeurs héritées des parents c^U. Le premier concept c_4 est le top du treillis et n'a pas donc de parents, son vecteur reste vide. c_4 est le parent de c_6, donc $v(c_6)$ n'est pas modifié. Le vecteur requête du concept c_2 construit dans la boucle de la ligne 3 ($v(c_2) = 00010$), est complété par le vecteur requête du parent c_8, $v(c_8) = 00101$ et deviens $v(c_2) = 00111$.

c	T	Q	$v(c)$	c^U	$t(c)$	P	$\lvert E^r(c)\rvert$	G
c_4		q_1, q_2, q_3, q_4, q_5	00000				0	
c_6	c_6	q_2, q_3, q_4, q_5	10000	c_4	q_1	c_6	0	
c_8	c_6, c_8	q_2, q_4	00101	c_4	q_3, q_5	c_6, c_8	0	
c_{11}	c_6, c_8	q_2, q_4	00000	c_4		c_6, c_8	0	
c_{12}	c_6, c_8	q_2, q_4	00000	c_4		c_6, c_8	0	
c_{13}	c_6, c_8	q_2, q_4	10000	c_6, c_{11}		c_6, c_8	0	
c_{15}	c_6, c_8	q_2, q_4	00000	c_{11}		c_6, c_8	0	
c_2	c_6, c_8, c_2	q_2	00111	c_8	q_4	c_6, c_8, c_2	0	
c_9	c_6, c_8, c_2	q_2	00101	c_8, c_{12}		c_6, c_8, c_2	1	c_8
c_5	c_6, c_8, c_2, c_5		01000	c_2, c_3	q_2	c_6, c_8, c_2, c_5	1	c_8, c_5
c_{10}	c_6, c_8, c_2, c_5		10101	c_8, c_6		$c_6, c_8, c_2, c_5, c_{10}$	1	c_8, c_5, c_{10}
c_{16}	c_6, c_8, c_2, c_5		10000	c_{13}		$c_6, c_8, c_2, c_5, c_{10}$	0	c_8, c_5, c_{10}
c_{17}	c_6, c_8, c_2, c_5		10000	c_{13}, c_{15}		$c_6, c_8, c_2, c_5, c_{10}$	0	c_8, c_5, c_{10}
c_0	c_6, c_8, c_2, c_5		00111	c_2, c_9		$c_6, c_8, c_2, c_5, c_{10}$	1	c_8, c_5, c_{10}, c_2
c_3	c_6, c_8, c_2, c_5		10111	c_2, c_{10}		$c_6, c_8, c_2, c_5, c_{10}, c_3$	1	$c_8, c_5, c_{10}, c_2, c_3$
c_7	c_6, c_8, c_2, c_5		11000	c_{17}, c_5		$c_6, c_8, c_2, c_5, c_{10}, c_3, c_7$	1	$c_8, c_5, c_{10}, c_2, c_3, c_7$
c_{14}	c_6, c_8, c_2, c_5		10000	c_{12}, c_{16}		$c_6, c_8, c_2, c_5, c_{10}, c_3, c_7$	1	$c_8, c_5, c_{10}, c_2, c_3, c_7, c_6$
c_{18}	c_6, c_8, c_2, c_5		10000	c_{16}, c_{17}		$c_6, c_8, c_2, c_5, c_{10}, c_3, c_7$	1	$c_8, c_5, c_{10}, c_2, c_3, c_7, c_6$
c_1	c_6, c_8, c_2, c_5		11111	$c_0, c_3, c_{14}, c_{18}, c_7$		$c_6, c_8, c_2, c_5, c_{10}, c_3, c_7, c_1$	0	$c_8, c_5, c_{10}, c_2, c_3, c_7, c_6$

Tableau 3.3: Trace de l'execution de l'algorithme 3

Le test de la ligne 8 identifie les concepts projections. Un concept c est projection si son vecteur $v(c)$ possède plus de 1s que tout vecteur d'un successeur immédiat (voir la propriété B.5). Le premier concept, c_4, n'est pas projection, car $|v(c_4)| = 0$. De la même façon, c_6 comparée individuellement a chacun de ses parents, nous avons $|v(c_6)| > |v(c_4)|$, donc c_6 est une projection.

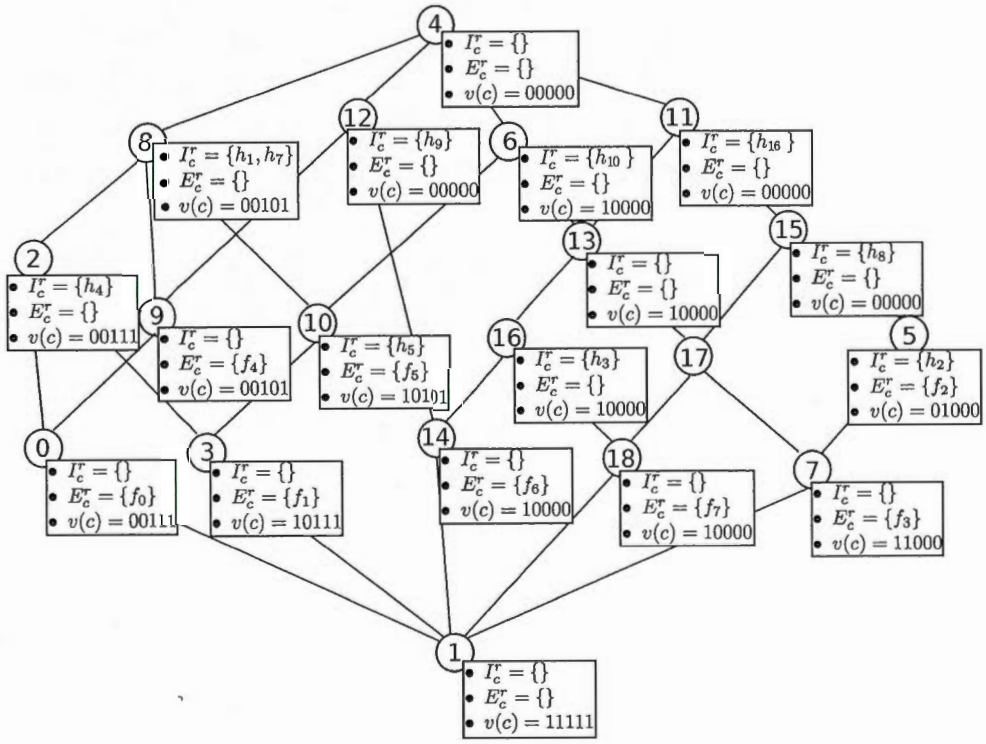


Figure 3.1: Treillis de concept avec les vecteurs requêtes de chaque concept

L'extension réduite d'un concept c, notée $E^r(c)$ est composée des flots qui existent dans $E(c)$, mais qui n'existent pas dans ces prédécesseurs ($E^r(c) = E(c) \setminus \bigcap_{\bar{c} \in c^L} E(\bar{c})$). Par exemple $E^r(c_5) = \{f_2\}$, puisque $E(c_5) = \{f_2, f_3\}$ et $E(c_7) = \{f_2\}$. Remarquons que pour un flot f, son concept flot $c = (f'', f')$ a pour extension réduite $E^r(c) = \{f\}$ ou encore $|E^r(c)| = 1$. Cette propriété est exploitée par le test de la ligne 8 pour identifier les concepts flots. Rappelons que les concepts flots sont c_9, c_5, c_{10}, c_0, c_3, c_7, c_{14} et c_{18}. Notons que cette propriété est une particularité des contextes FHM, en général, $E^r(c)$ peut avoir plusieurs éléments, le test serait donc $E^r(c) \neq \varnothing$.

Pour chaque concept flot c découvert, les instructions de la boucle *for* de la ligne 11 trouvent à quel concept projection p associer son flot f. Il s'agit du concept p qui possède

le même vecteur requête que c (ligne 12). Par exemple, le concept projection qui a le même vecteur requête que c_9 est c_8 (voir figure 3.1). Donc c_8 est un concept base, plus particulièrement, $g(f_4) = c_8$. De même, le concept projection p tel que $v(c_5) = v(p)$ est c_5, donc c_5 est rajouté à G, de plus $g(f2) = c_5$. L'ensemble des concepts bases $G(Q)$ est $\{c_2, c_3, c_5, c_6, c_7, c_8, c_{10}\}$.

3.5 L'affectation des compteurs et calcul des valeurs des requêtes

La partition optimale des flots est l'ensemble des extensions des concepts bases (voir section 3.3.2). Pour effectuer la tâche de surveillance, à chaque $c \in G(Q)$ est attribué un compteur, ce qui signifie qu'un flot f est affecté au compteur (unique) de son $\mu_Q(f)$ (lorsque ce concept est différent du $\top$).

La mesure exacte d'une requête q est obtenue par le biais de sa cible $\gamma(q) \in T(Q)$, c'est la somme des valeurs des compteurs affectés aux flots dans l'extension $E(\gamma(q))$. Le calcul est fait une seule fois en additionnant tous les compteurs correspondant aux concepts de base qui sont des sous-concepts de $\gamma(q)$ puis nous ajoutons le compteur local de $\gamma(q)$ dans le cas où il est lui-même un concept-base (si $(\gamma(q) \in G(Q))$. Par exemple, pour $c_5 = \gamma(q_2)$, $E(c_5)\{f_2, f_3\}$, la somme comprend le compteur de c_7 ($c_7 = \mu_Q(f_3)$) et son propre compteur (comme $c_5 = \mu_Q(f_2)$). De même, pour $c_6 = \gamma(q_1)$, $E(c_6) = \{f_1, f_3, f_5, f_6, f_7\}$, la somme comprend les compteurs des concepts bases $c_3 = \mu_Q(f_1)$, $c_7 = \mu_Q(f_3)$, $c_{10} = \mu_Q(f_5)$, $c_6 = \mu_Q(f_6) = \mu_Q(f_7)$, remarquons que les mesures de f_6 et f_7 sont faites par le même compteur associé à c_6.

3.6 L'actualisation de l'affectation des compteurs

Comme le treillis des flots évolue, l'ensemble des réponses et la partition des flots évoluent aussi. L'objectif de cette section est de décrire l'algorithme de mise à jour incrémentale des nouveaux concepts cibles, projections et bases lors de l'ajout d'un nouveau flot. Dans la suite, T_n, P_n et G_n dénotent les ensembles T, P et G, respectivement après mise à jour avec le nouveau flot f_n.

Le travail principal consiste à établir le statut de chaque nouveau concept c_n et son géniteur c et d'identifier le concept base du flot f_n. Le changement qui s'est produit après l'ajout de f_n, est que soit f_n est ajoutée à des concepts existants ou de nouvelles intersections des intensions de certains concepts avec f'_n ont généré de nouveaux concepts (voir section 2.5.2). Certaines requêtes q peuvent changer de cible, mais toutes les q ne se déplacent pas nécessairement, car

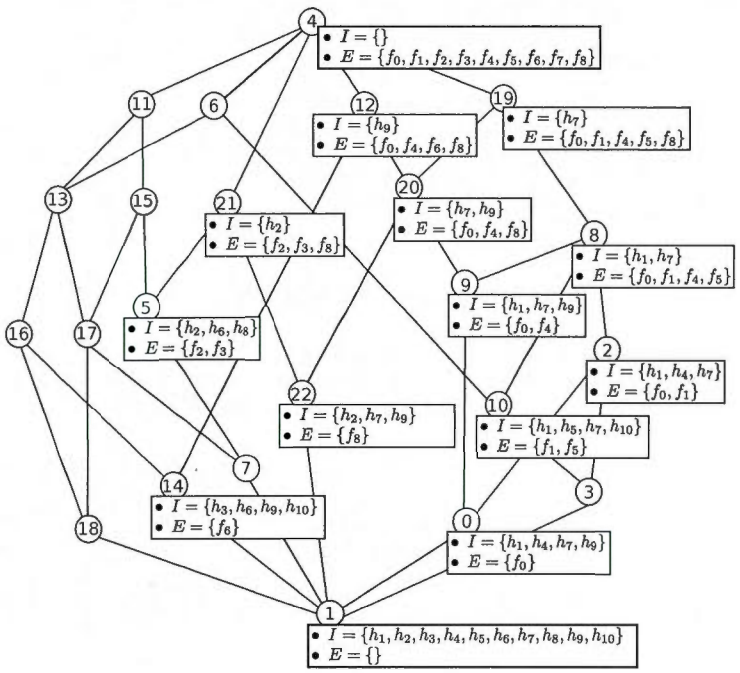


Figure 3.2: Treillis de concepts eFHM étendu par ajout du flot f_8 (repris)

de nouvelles intersections d'intensions sont apparues, dans ce cas, le déplacement de leurs cibles se fait du géniteur vers le nouveau. Le test de statut de cibles se réduit donc aux nouveaux concepts et à leurs géniteurs. Pour un géniteur cible, on teste seulement s'il doit prendre son statut de cible. Un concept projection de P_n, qui est la borne inférieure de certains concepts cibles T_n, peut soit se déplacer d'un concept géniteur vers le nouveau, de la même façon que les concepts cibles ont évolués, soit être nouveau concept qui devient projection, car il est une nouvelle borne inférieure de certains concepts cibles de T_n. Puisque G_n est sous-ensemble de P_n, le changement de statut de base suit le changement de statut de projection.

La méthode UpdateStatus (détaillée par l'algorithme 4) se décompose en trois blocs: établissement du statut (cible, projection, base, aucun) de chaque concept c_n nouvellement crée par l'algorithme 2 et mise à jour du statut de son géniteur (lignes 1-15), identification du concept base du flot f_n nouvellement ajouté (ligne 16) et actualisation de G (ligne 18). Le tableau 3.4 présente la trace de l'exécution de cette méthode sur les concepts c_8, c_9, c_5 et c_1 géniteurs des nouveaux concepts c_{19}, c_{20}, c_{21} et c_{22}, respectivement.

Algorithm 4: UpdateStatus : Mise à jour du demi-treillis

input/output: Les concepts $(c_n, c) \in N$ (nouveau et son géniteur)
Ensembles de concepts (T, P, G)

1 **foreach** $(c, c_n) \in N$ **do**
2 **foreach** $q_i \in t(c)$ **do**
3 **if** $q_i \subseteq I(c_n)$ **then**
4 $t(c_n) \leftarrow t(c_n) \cup \{q_i\}$; $t(c) \leftarrow t(c) - \{q_i\}$;
5 $v(c_n)[i] \leftarrow 1$;
6 **if** $t(c_n) \neq \varnothing$ **then** $T \leftarrow T \cup \{c_n\}$; // Mise à jour de T
7 **if** $t(c) = \varnothing$ **then** $T \leftarrow T - \{c\}$;
8 $v(c_n) \leftarrow v(c_n) \cup \bigcup_{\overline{c} \in c_n^U} v(\overline{c})$;
9 **if** $|v(c_n)| > \max_{\overline{c} \in c_n^U} (|v(\overline{c})|)$ **then**
10 $P \leftarrow P \cup \{c_n\}$; // Mise à jour de P
11 **if** $|v(c)| = |v(c_n)|$ **then**
12 $P \leftarrow P - \{c\}$;
13 **if** $g(c) \neq \varnothing$ **then**
14 $g(c_n) \leftarrow g(c)$; $g(c) \leftarrow \varnothing$;
15 $G \leftarrow G \cup \{c_n\} - \{c\}$; // Mise à jour de G
16 $c_p \leftarrow Lookup(P, v(\mu(f_n))$; // Calcul de la base de f_n
17 $g(c_p) \leftarrow g(c_p) \cup \{f_n\}$;
18 $G \leftarrow G \cup \{c_p\}$; // Ajout sans test d'appartenance

Lors de l'ajout d'un nouveau flot f_n, des requêtes q peuvent changer de concept cible. Le déplacement de la cible d'une requête ne peut se faire que d'un géniteur vers un nouveau concept (voir la propriété B.14). De même, seuls les nouveaux concepts peuvent devenir de nouvelles projections, possiblement en "évinçant" de l'ensemble P leurs géniteurs respectifs. Le test de projection se réduit donc aux nouveaux concepts et leurs géniteurs (lignes 2-7). Les vecteurs requêtes des anciens concepts ne changent pas (voir la propriété B.14), nous devons donc calculer les vecteurs requêtes uniquement pour les nouveaux concepts (ligne 8). Les concepts projections P_n, définis comme bornes inférieures des concepts cibles T_n peuvent soit (1) se déplacer d'un ancien concept vers un nouveau soit (2) se créer si une nouvelle borne inférieure des concepts cibles apparait (voir les propriétés B.15 et B.16). Le principe est donc de vérifier si le nouveau

concept est projection (ligne 9) et si de plus son géniteur a le même vecteur requête (ligne 11) donc il s'agit d'une projection qui s'est déplacée "vers le haut", il faut donc retirer le géniteur de P_n (ligne 12). G_n est un sous ensemble de P_n, donc lorsqu'on déplace une projection, il faut déplacer en même temps les flots qui y sont basés vers le nouveau concept c_n (lignes 14-15), ceci met à jour les bases des flots de $\mathcal{F}$, il reste, pour finaliser à trouver le concept base de f_n (ligne 16).

Avant exécution				Après exécution					
c	$t(c)$	c_n	$I(c_n)$	$t(c_n)$	$t(c)$	T_n	c_n^U	P_n	G_n
c_8	q_3, q_5	c_{19}	h_7	q_5	q_3	$T \cup \{c_{19}\}$	c_4	$P \cup \{c_{19}\}$	$G \cup \{c_{19}\}$
c_9		c_{20}	h_7, h_9			$T \cup \{c_{19}\}$	c_{12}, c_{19}	$P \cup \{c_{19}\}$	$G \cup \{c_{19}\}$
c_5	q_2	c_{21}	h_2		q_2	$T \cup \{c_{19}\}$	c_4	$P \cup \{c_{19}\}$	$G \cup \{c_{19}\}$
c_1		c_{22}	h_2, h_7, h_9			$T \cup \{c_{19}\}$	c_{20}, c_{21}	$P \cup \{c_{19}\}$	$G \cup \{c_{19}\}$

Tableau 3.4: Trace de l'execution de l'algorithme 4

Le test de la ligne 3 identifie les requêtes pour lesquels c_n est cible. Pour le concept c_8, géniteur de c_{19}, $t(c_8) = \{q_3, q_5\}$. La boucle *for* (ligne 2) va passer sans effet pour $q_3 = \{h_1\}$ puisque $h_1 \nsubseteq I(c_{19}) = \{h_7\}$. Par contre $q_5 = \{h_7\} \subseteq I(c_{19})$, donc q_5 va être "reciblée" vers $c_n = c_{19}$ (ligne 4) et $T_n = T \cup c_{19}$ (ligne 6). Évidemment, c_8 reste dans T_n, car $t(c_8) = \{q_3\}$. Pour c_9 et c_1, $t(c_9) = t(c_1) = \emptyset$ (ligne 7), T reste stable parce que $c_9 \notin T$ et $c_1 \notin T$.

La mise à jour de $v(c_n)$ et le test de projection (lignes 8-9) suivent exactement les instructions de l'algorithme 3 (lignes 7-8), ici, le test valide que c_{19} est une projection. Par la suite, il faut voir si la nouvelle projection a masqué son géniteur (ligne 11), dans le cas échéant, retirer le géniteur de l'ensemble des projections (ligne 12) et déplacer les flots qui sont basés sur lui vers le nouveau concept (lignes 13-15), dans le cas de c_{19}, il ne s'agit pas d'une projection qui s'est déplacée ($v(c_8) = 00101 \neq v(c_{19}) = 00001$), donc c_8 conserve son statut de projection.

Enfin, la maintenance des compteurs est reprise (lignes 16-19): le concept-base de f_n est identifié par le fait qu'il est la projection $c_p \in P$ possédant le même vecteur de requête que le concept-flot $\mu(f_n)$ (ligne 16). Dans notre exemple, $v(\mu(f_8)) = v(c_{22}) = 00001$ est le même que le vecteur-requête de la projection c_{19}. Le nouveau flot f_8 a donc comme concept-base le concept c_{19} (ligne 18).

Pour résumer la restructuration: La nouvelle liste des concepts-cibles est $T_n = T \cup \{c_{19}\}$, les nouvelles projections $P_n = P \cup \{c_{19}\}$, et les nouvelles bases $G_n = G \cup \{c_{19}\}$.

3.7 Conclusion

Dans ce chapitre, nous avons introduit des algorithmes originaux de construction et d'actualisation du demi-treillis des projections qui identifie la partition optimale des flots à superviser par les compteurs. Les algorithmes que nous avons introduits sont utilisés au niveau du collecteur FLOWME et permettent essentiellement de trouver les partitions minimales des flots à superviser pour répondre à un ensemble de requêtes des utilisateurs. La conception et l'implémentation de ce collecteur sont détaillées dans le chapitre suivant.

CHAPITRE IV

LA CONCEPTION ET L'IMPLÉMENTATION DE FLOWME

4.1 Introduction

Les outils d'observation de trafic requièrent une grande flexibilité afin de permettre la couverture de tous les types d'application et des requêtes d'observation des administrateurs. La solution de surveillance FlowME exploite les associations qui existent entre les requêtes utilisateurs et les descripteurs de flots traités par le nœud de communication pour minimiser le besoin en classification et mise à jour des compteurs. Dans ce chapitre, nous proposons une implémentation de FlowME sur un nœud de communication OpenFlow. Nous avons réalisé cette implémentation avec un collecteur qui maintient les algorithmes de treillis et met à jour l'affectation des entrées de flots aux compteurs et d'une routine d'incrémentation des compteurs au niveau du chemin de traitement des paquets du nœud de communication.

4.2 Le flux global de la solution

Le collecteur FlowME maintient les algorithmes de treillis et met à jour l'affectation des entrées de flots aux compteurs. Il interagit avec les tables des entrées de flots matérielles et les mémoires de compteurs à travers le processeur hôte (*host*). La figure 4.1 illustre l'architecture de notre implémentation. Le nœud de communication est constitué principalement d'un processeur réseau (NPU), un processeur *host*, des tables de recherche et des mémoires des compteurs.

Le NPU effectue le traitement des paquets et incrémente les compteurs. Les compteurs matériels sont implémentés sur des mémoires SRAM et RLDRAM. Dans notre implémentation, nous utilisons la SRAM pour les 8192 premiers compteurs et la RLDRAM pour le reste. Le bloc matériel de statistiques (statistics block) est un ordonnanceur pour les opérations sur les

compteurs matériels. Les opérations de mise à jour et de lecture viennent du NPU ou de le *host*. Les tables des entrées de flots sont implémentées avec des structures de recherche TCAM et des tables de hachage. Le contrôleur de TCAM et l'Entries Pusher sont deux programmes qui sont exécutés par le *host* et qui permettent de modifier le contenu de la TCAM et des tables de hachage. Le programme Counter Puller de le *host* permet au collecteur de lire et initialiser les compteurs matériels.

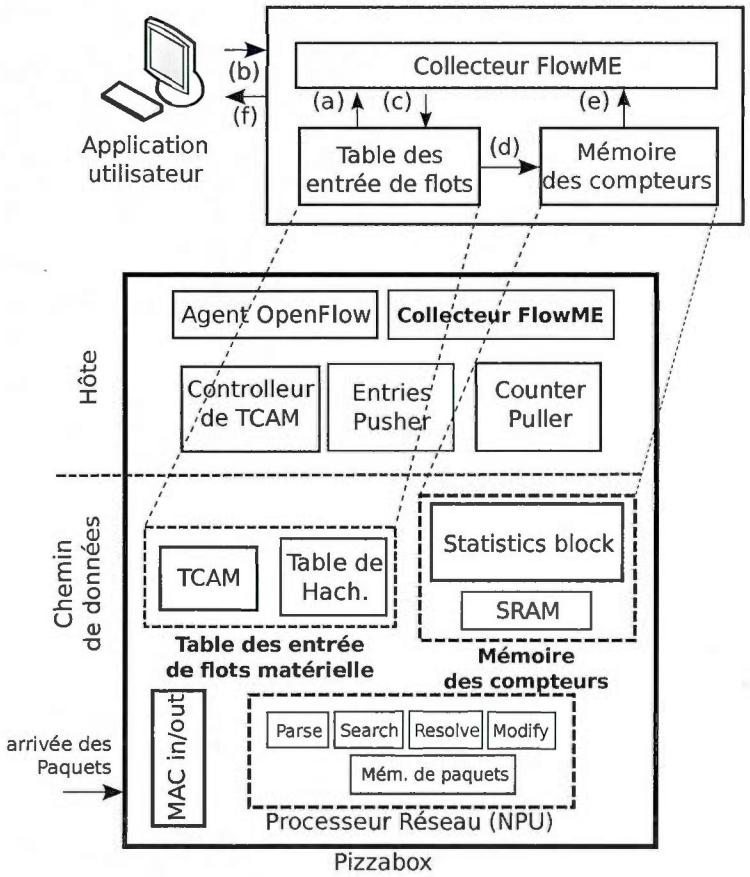


Figure 4.1: Architecture de l'implémentation de FLOWME

Le collecteur FLOWME, reçoit l'ensemble des entrées de flots $\mathcal{F}$ (a) installées dans la table matérielle des entrées de flots. Le collecteur prend en entrée aussi les requêtes utilisateur Q (b). Le collecteur fait appel ensuite aux algorithmes que nous avons présentés dans le chapitre 2 et 3 afin de calculer, maintenir la partition optimale des entrées de flots et trouver les associations flots à compteurs (c).

Le chemin de données reçoit le trafic, identifie l'appartenance des paquets aux flots et incrémente les compteurs matériels associés (d). Le collecteur FLOWME lit ensuite les valeurs des compteurs (e) et calcule les réponses aux requêtes (f). Les résultats sont renvoyés aux applications qui ont généré les requêtes.

Nous avons expérimenté FLOWME avec une variété d'entrées et de distributions de requêtes. Notre implémentation du chemin de traitement des paquets du nœud de communication est basée sur l'implémentation d'OPENFLOW sur le processeur réseau NP-4 d'EZchip [26]. Les modifications apportées à cette implémentation pour supporter FLOWME sont minimales. L'ajout de la référence du compteur à incrémenter dans le résultat de chaque entrée de flot et l'ajout d'une routine d'incrémentation des compteurs. Ces modifications peuvent donc être reproduites facilement sur d'autres types d'équipements réseau.

4.3 Le collecteur FLOWME

Le collecteur FLOWME effectue essentiellement trois tâches : (1) la construction et la maintenance des structures de treillis (2) l'affectation des compteurs aux entrées de flots et (3) le calcul des valeurs des requêtes utilisateur.

4.3.1 Le module de construction et maintenance du treillis

Le module de construction de treillis est responsable de la construction du treillis des flots de départ, de la maintenance du treillis des flots lors de l'ajout/suppression de flots et de l'identification des groupes des flots à superviser. Nous avons établi nos algorithmes en nous basant sur la suite FCA Coron [20] qui donne plusieurs outils d'exploration des treillis de concepts. La construction du treillis se fait une seule fois lors du lancement du système. Par la suite, les algorithmes de maintenance permettent de mettre à jour le treillis des flots sans reconstruction totale. Les groupes des flots à superviser sont déterminés au moment où les concepts-bases sont produits. L'extension de chaque concept base correspond à un groupe d'entrées de flots à superviser. Dans l'exemple de départ, les concepts-bases c_2, c_3, c_5, c_7, c_8, c_{10} et c_6 donnent les groupes de flots $\{f_1\}$, $\{f_2\}$, $\{f_3\}$, $\{f_4\}$, $\{f_5\}$, $\{f_6, f_7\}$, respectivement.

Les valeurs des requêtes sont calculées à partir des compteurs des groupes des flots. Premièrement, les valeurs des compteurs sont reportées aux concepts-bases auxquels ils sont attachés. Ensuite, la valeur d'un concept cible est obtenue en sommant les valeurs des concepts-

bases qui lui sont inférieurs (voir section 3.5). Par exemple, la requête q_1 a comme concept cible c_6. Les sous-concepts base du concept c_6 sont c_6 lui-même, c_{10}, c_3 et c_7 qui sont affectées aux groupes de flots $\{f_6, f_7\}$, $\{f_5\}$, $\{f_1\}$ et $\{f_3\}$ respectivement. La valeur de q_1 est la somme des compteurs associés à ces groupes de flots.

4.3.2 Le mécanisme d'affectation des compteurs aux entrées de flots

Le mécanisme d'affectation des compteurs aux entrées de flots associe au même compteur les flots appartenant à la même extension d'un concept-base. Dans notre implémentation, nous plaçons l'adresse du compteur à incrémenter dans tous les flots appartenant au même groupe.

Dans le cas simple de l'existence d'une seule table de flots à superviser dans le nœud de communication, l'incrémentation du compteur associé au flot se fait directement après le passage par cette table.

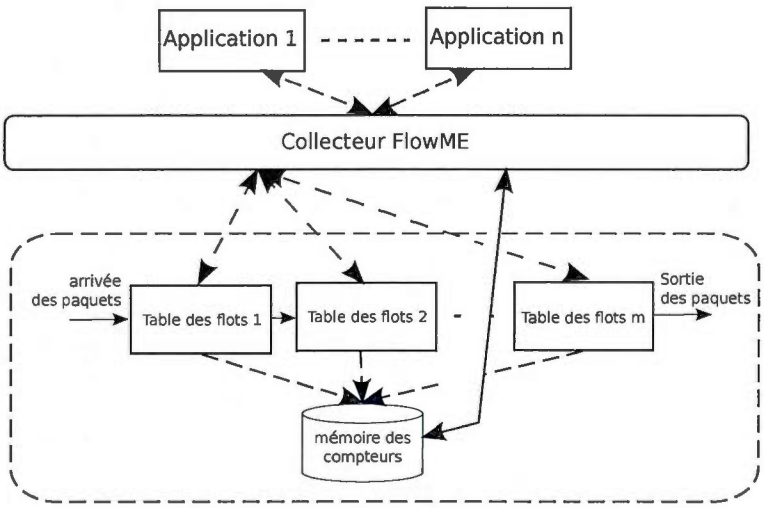


Figure 4.2: Implémentation sur un nœud de communication basé sur OpenFlow

La taille d'une table de flots évolue exponentiellement avec le nombre des champs d'entête de ces entrées. Par exemple si la table est constituée de d champs de taille W, elle requiert un espace mémoire de $(2W-2)^d$ [25]. Le pipeline de tables des flots décompose la table de flots original et réduit donc d, c'est une stratégie largement utilisée dans les nœuds de communication actuels pour éviter l'explosion cartésienne de l'espace mémoire.

Dans notre solution, chaque flot en mémoire peut être réparti sur un ensemble de tables en pipeline (voir figure 4.2). L'identifiant du compteur de flot est construit au fur et à mesure du parcours des tables de flot du pipeline. Par exemple, dans le cas d'un pipeline de n tables, chaque flot est décomposé en n parties $f = (e_1, ..., e_n)$. Le compteur du flot f, est construit à la fin du pipeline par concaténation. Par exemple, avec une décomposition de 3 tables, si nous voulons affecter le compteur d'adresse 0xAABBCC au flot f, à la première table on identifie e_1 et on récupère 0xAA comme adresse partielle du compteur, la deuxième table identifie e_2 et on récupère 0xBB, la troisième table identifie e_3 et on récupère 0xCC, à la fin du pipeline on concatène les valeurs récupérées et on incrémente le compteur du flot f.

4.4 Le traitement des paquets

Le nœud de communication doit supporter trois opérations : l'affectation des compteurs aux flots, l'incrémentation et la lecture des compteurs des groupes de flots. Dans la suite, nous montrons comment ses opérations sont réalisées par le pipeline du traitement des paquets.

4.4.1 La table des entrées des flots

La table des entrées de flots est utilisée dans le nœud de communication à la fois pour identifier les paquets pour les opérations de traitement de paquet et pour effectuer la tâche de surveillance du trafic des flots. Le tableau 4.1 présente des champs d'en-têtes des entrées de flots de l'exemple de départ. L'entrée de flot standard du protocole OpenFlow définit 14 champs d'en-têtes (*match fildes*). Les entrées de flots correspondants à notre exemple définissent les champs d'en-têtes MAC source, MAC destination, IP source, IP destination et champs destination du protocole UDP ou TCP. Les champs d'en-têtes restants ne sont pas spécifiés et leurs valeurs seront ignorées par l'algorithme de classification.

Les entrées de flots gardent la même structure mentionnée dans la spécification du protocole OpenFlow. La modification que nous avons introduite est au niveau du champ "compteurs" de l'entrée de flot. Au lieu de mettre le résultat de la mesure du flot (nombre de paquets et taille totale) dans la structure de données du flot, nous remplaçons le champ compteur du flot par un pointeur dirigé vers le compteur du groupe de flots correspondant placé dans une mémoire externe. Dans notre exemple de base, les flots f_6 et f_7 pointent vers le compteur commun qui reflète l'extensions du concept base c_6.

Flot	Port	MAC src	MAC dst	IPv4 src	IPv4 dst	L4 dst
f_0	1	*:45:f5:01	*	10/8	132.208.130.1	*
f_1	1	*:45:f5:01	*:45:f5:12	10/8	*	21
f_2	2	*	*	132.208.130/32	10/8	*
f_3	2	*	*	132.208.130/32	10/8	21
f_4	1	*	*	10/8	132.208.130.1	*
f_5	1	*	*:45:f5:12	10/8	*	21
f_6	3	*	*	132.208.130/32	132.208.130.1	21
f_7	3	*	*	132.208.130/32	10/8	21

Tableau 4.1: Table des entrées de flots de l'exemple de départ

La table de flots est implémentée en utilisant une TCAM (Ternary Content-Addressable Memory) et une table de hachage. La TCAM classifie le flot et renvoi l'identifiant du flot trouvé vers la deuxième table. Rappelons que la TCAM est une mémoire adressée par contenu, donc le résultat de la classification est l'adresse du flot dans la TCAM. La deuxième table contient les instructions à exécuter sur le paquet ainsi que l'adresse du compteur à incrémenter.

4.4.2 Le pipeline du traitement des paquets

Les opérations de traitement de paquet que nous avons implémentés sont déployés sur les quatre étages du processeur réseau (voir la figure 4.1).

Le pipeline du traitement des paquets reprend l'implémentation de base [9] et lui ajoute la gestion des compteurs des groupes de flots. Le premier étage analyse les champs d'en-têtes pertinents du paquet (*parsing*) et construit une clé de 14 champ. Cette clé est envoyée à l'étage suivant pour identifier à quel flot le paquet appartient. Le deuxième étage (*search*) réalise la classification du paquet, en effet, il passe la clé qu'il a reçue à la TCAM, et si un résultat est trouvé, l'indice de l'entrée de flot est envoyé à une table de hachage pour une deuxième recherche afin de trouver les instructions du traitement du paquet et le compteur à incrémenter.

Le paquet est ensuite envoyé au troisième étage qui réalise la résolution du paquet (*resolving*). La résolution consiste à préparer les actions à effectuer sur le paquet et incrémenter le compteur correspondant au flot. L'opération d'incrémentation du compteur est effectuée en envoyant une commande au bloc matériel des statistiques. Le bloc matériel des statistiques du processeur réseau (*statistics block*) ordonnance les demandes de lecture et d'écriture dans la

mémoire des compteurs. Son rôle est crucial pour réduire l'impact de la gestion des compteurs sur les performances du paquet. Le processeur réseau envoie une commande d'incrémentation et sans attendre de retour, il continue le traitement du paquet.

Le dernier étage du traitement de paquet (*modifying*) permet d'appliquer les actions et les décisions de l'étage de résolution. Il reçoit en entrée une liste des tâches à effectuer sur le paquet. Après application des actions, le paquet est mis dans la queue d'une interface de sortie.

4.5 Conclusion

Dans ce chapitre, nous avons établi la conception de FLOWME et son implémentation sur un nœud de communication physique. Notre solution exploite les associations entre les flots traités par le nœud de communication et les requêtes utilisateurs pour donner une affectation optimale des compteurs. Nous prévoyons donc une réduction significative du nombre des compteurs utilisés et par la suite les coûts liés au stockage et à la gestion des compteurs dans le nœud de communication. Dans le chapitre suivant, nous allons tester notre solution dans des conditions de trafic réel pour valider cette hypothèse.

CHAPITRE V

L'ÉVALUATION DE L'APPROCHE FLOWME

5.1 Introduction

Dans ce chapitre nous évaluons notre stratégie de surveillance du trafic réseau sur le nœud de communication dans des conditions de trafic réel. L'implémentation de FLowME sur un nœud de communication basé sur OpenFlow sera étudiée selon deux axes : le coût de mémoire, exprimé en terme de nombre de compteurs gérés, et l'effort de traitement requis pour la génération de treillis de concepts.

Le banc d'essai de FlowME que nous avons construit comporte un nœud de communication OpenFlow qui supporte les compteurs par flot, un générateur d'entrées de flots, un collecteur et un générateur de requêtes. Les expérimentations sont effectuées avec différentes entrées de flots et différentes distributions de requêtes.

5.2 La conception de l'environnement de l'évaluation

Dans cette section, nous décrivons l'environnement expérimental utilisé pour l'évaluation de FlowME. Nous commençons par décrire le nœud de communication qui implémente FlowME, ensuite, les entrées du système, à savoir, les paquets, les entrées de flots et les requêtes utilisateur.

5.2.1 La génération des paquets et des entrées de flots

Notre générateur de flots génère des entrées de flots de 12 champs. Le générateur est basé sur l'outil Frug (Flexible Rule Generator) [10]. Frug est un outil d'évaluation des algorithmes de classification. Il génère des entrées de flots OpenFlow à partir des distributions de champs

d'en-têtes prédéfinies.

Nous avons extrait la distribution de champs d'entêtes à partir de distributions de paquets. Les paquets qui sont utilisés dans nos expérimentations sont extraits des différentes traces obtenues du site packetlife.net. Ces traces sont particulièrement intéressantes puisque les champs d'en-têtes sont de plusieurs types : MAC, VLAN, IP et champs de protocoles de transport, ainsi de suite. Nous avons choisi dans notre expérimentation une trace de paquets qui nous fournit un total de 12 champs d'en-têtes OPENFLOW standards. Notez que les traces fournies par d'autres sites tels que caida.org nous permettent de générer au plus des entrées de flots de 5 champs.

La table 5.1 montre un ensemble de champs d'en-têtes et la valeur de leurs distributions. Seuls les champs avec des densités $\geqslant 3\%$ sont repris dans la table. Il est à noter que la génération des adresses IP source et destination se base sur deux paramètres : la distribution des préfixes et la distribution des longueurs des préfixes.

Tableau 5.1: Distribution des champs d'entêtes de trace de paquets

Champs d'entêtes	Distribution
MAC src	00:40:05(39%), 08:00:07(13%),
MAC dst	00:60:08(33%), FF:FF:FF(37%),
Ethertype	0x8100(98)%
VLAN id	32(56%), 104(17%), 108(4%), 6(6%)
IP protocol	0x06(80%), 0x11(6%), 0x01(13%)
TOS	0(96%), 192(3%)
L4 src port	2212(41%, 1815(26%), 2388(11%), 8(4%)
L4 dst port	1815(53%), 2212(18%),

5.2.2 La génération des requêtes utilisateurs

Le deuxième générateur produit les requêtes utilisateurs. Chaque requête générée porte sur un ensemble d'entrées de flots. Plus une requête contient de champs d'en-têtes non renseignés (méta caractère *), plus elle porte sur un nombre supérieur d'entrées de flots. Dans notre étude expérimentale, nous générons les requêtes des utilisateurs avec la même distribution des champs d'en-têtes que celle des entrées de flots. Ceci est achevé en sélectionnant un sous-ensemble de

flots et puis en masquant aléatoirement des champs d'en-têtes par une distribution prédéfinie. Un pourcentage spécifique est associé pour chaque champ d'en-tête.

À cette fin, on extrait tout d'abord un ensemble de n entrées de flot de $\mathcal{F}$, puis on insère un pourcentage spécifique de caractères génériques dans chaque champ d'en-tête, ce qui donne un ensemble de requêtes n. L'avantage de cette méthode est que nous sommes certains de couvrir au moins une entrée de flot pour chaque requête.

5.3 Le temps d'établissement du treillis

Dans cette section, nous évaluons les temps de création et de mise à jour des structures de treillis par ajout de nouvelles entrées.

5.3.1 Le temps de création du treillis

La création du treillis initial n'est effectuée qu'au démarrage du système. Par la suite, les algorithmes de maintenance du treillis permettent sa mise à jour. La construction se découpe en deux phases principales : la construction du treillis des entrées des flots aux champs d'en-têtes et la construction du support de mesure. Le support de mesure est composé des trois ensembles de concepts T, P et G. La table 5.2 indique le temps de création du treillis et d'identification des ensembles T, P et G.

# flots	Distr.[1]	# requêtes	Temps (s)
1 000	90%	50	6
1 000	10%	500	8
5 000	90%	50	107
5 000	10%	500	146

Tableau 5.2: Le temps de création du treillis

[1]Distribution des champs non renseignées dans les requêtes utilisateurs

5.3.2 Le temps de mise à jour du treillis

La flexibilité de FLOWME est évaluée en mesurant le temps de mise à jour pour l'ajout d'une nouvelle entrée de flot. Les opérations principales sont la mise à jour du treillis, l'identification des concepts bases et l'extraction des partitions de flots. Dans notre expérience, 10.000 entrées de flots sont divisées en groupes de 100. Ces groupes d'entrées sont ensuite envoyés successivement au système. Cette méthode de construction incrémentale nous permet de caractériser l'évolution du treillis. La figure 5.1 indique le nombre d'entrées de flots ajoutées à chaque étape de la construction incrémentale du treillis. Par exemple, dans les premiers 8 s, 1000 flux sont ajoutés. Dans la moyenne des tests, il a fallu un total de 5 min 30 s pour arriver à construire le treillis d'une manière incrémentale et identifier les partitions de flots.

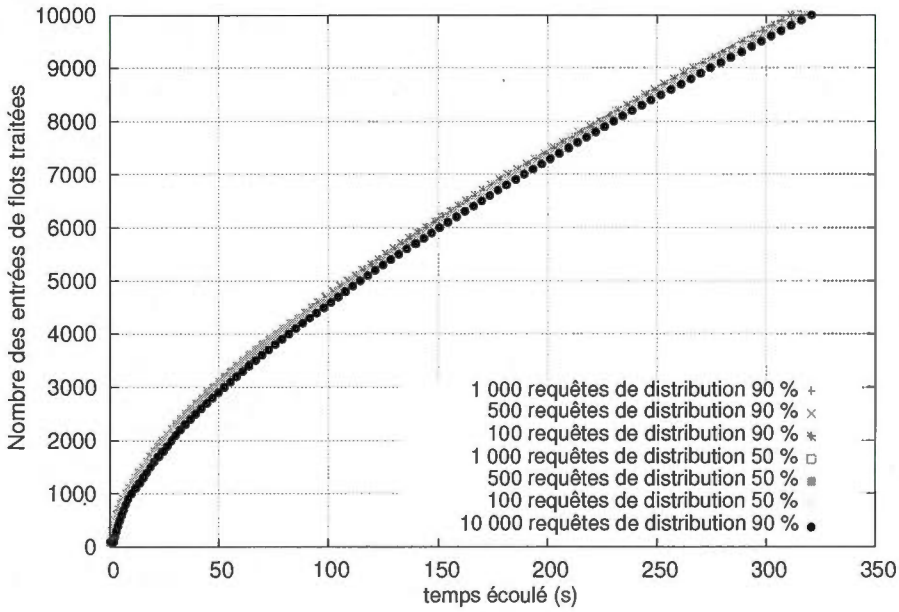
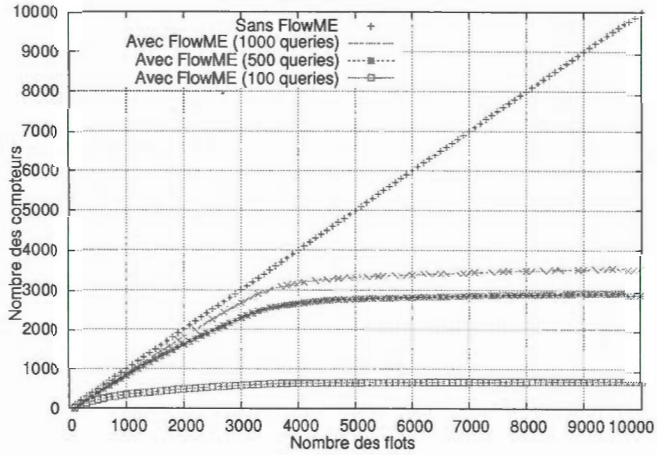


Figure 5.1: Le temps de construction incrémentale du treillis

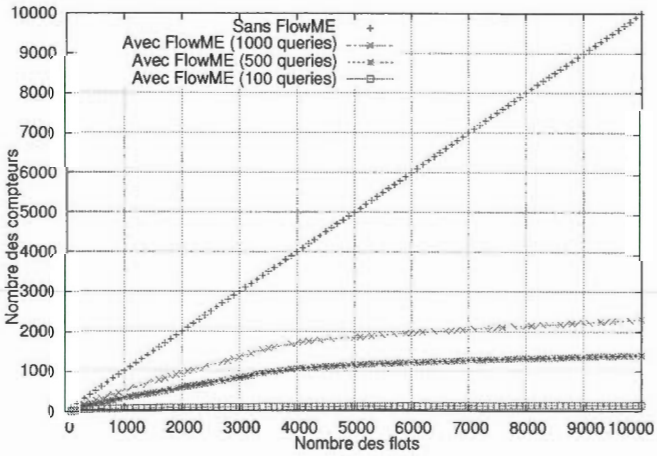
Un groupe de 100 entrées de flots est ajouté à chaque itération

5.4 L'évaluation de la réduction du nombre des compteurs

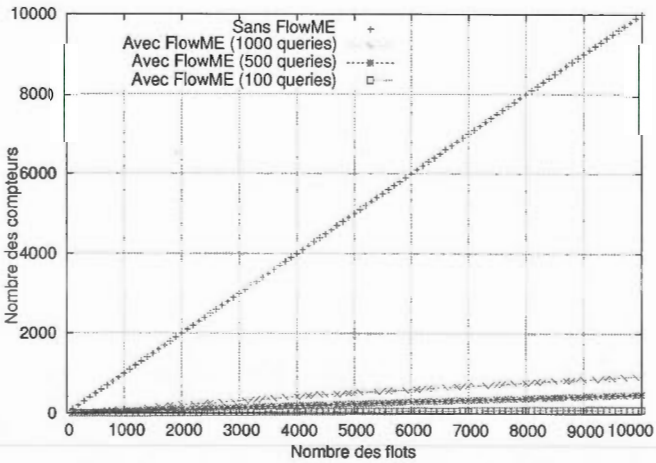
Les outils de surveillance du trafic par flot gèrent un compteur pour chaque flot de trafic traité par le système (l'ensemble $\mathcal{F}$) et rapportent les résultats de chaque flot à un collecteur



(a) 90% des champs d'entêtes non renseignées



(b) 50% des champs d'entêtes non renseignées



(c) 10% des champs d'entêtes non renseignées

Figure 5.2: Nombre des compteurs déployés pour plusieurs ensembles de requêtes avec différentes distributions de champs d'entêtes dans les requêtes.

centralisé. Dans ces systèmes, le nombre des compteurs N_c évolue linéairement avec $|\mathcal{F}|$ puisque, dans la pratique, pour chaque flot $f \in \mathcal{F}$, le système peut avoir besoin de différentes mesures sur le même flot, par exemple le nombre de paquets correspondant au flot et leur taille totale.

Notre solution repose sur des compteurs agrégés. La couverture des flots par les requêtes utilisateurs est le facteur déterminant du nombre des compteurs qui seront déployés. Les principaux variables sont le nombre des flots, la densité des champs d'en-têtes dans l'ensemble des flots, le nombre des requêtes utilisateur et la densité des champs d'en-têtes dans ces requêtes.

Soit M la taille de la mémoire disponible pour l'opération de surveillance du trafic et soit M_u le nombre des registres de compteurs gérés dans une période de mesure spécifique. M_u ne peut pas excéder le nombre d'entrées de flots installées dans le nœud de communication. Dans les systèmes de surveillance traditionnels, $M_u = |\mathcal{F}|$, en revanche, dans notre solution $M_u = |S(Q)| \leqslant |\mathcal{F}|$, avec $S(Q)$ la partition des flots associée à l'ensemble des requêtes Q. Le nombre des compteurs atteignant $|\mathcal{F}|$ constitue le pire des cas qui se produit uniquement si les intersections générées par Q décomposent $\mathcal{F}$ en singletons.

Nous avons déployé FLOWME avec différentes distributions de flots et de requêtes et nous avons observé le nombre de compteurs N_c.

L'évolution du nombre de compteurs à maintenir afin de répondre à un ensemble de requêtes des utilisateurs Q de taille N_Q est représentée dans la figure 5.2. Dans la figure 5.2a, les valeurs des champs des requêtes sont composées de 10% de valeurs renseignées et de 90% de valeurs non renseignées, alors que dans les figures 5.2a et 5.2c, les requêtes sont plus spécifiques avec 50% et 90% des valeurs renseignées, respectivement. La première expérience montre que, pour un $N_Q = 1000$, N_c est nettement inférieur au nombre des compteurs par flot qui est de 10.000 compteurs. N_c est entre 949 à 3555 compteurs, selon de la distribution des champs renseignée. D'une façon générale, on constate que lorsque les requêtes sont moins spécifiques, et couvrent plus d'entrées de flots, le nombre minimal d'intersections est plus élevé et donc le nombre des compteurs grandit.

Dans l'expérience 1 la figure 5.2a où $N_Q = 1000$, seuls 3555 compteurs sont rapportés en utilisant FLOWME, ce qui représente 43% de l'espace SRAM total réservé aux compteurs de statistiques. Nous avons distribué l'espace mémoire alloué aux compteurs par flot sur une mémoire SRAM de 8192 lignes et sur une mémoire RLDRAM de 16 millions de lignes. Cette distribution est particulièrement intéressante pour FLOWME, tant que M_u est inférieure à la

taille de la SRAM, l'accès mémoire des flots est de 2-3ns (SRAM), alors qu'il sera de 20-35ns (RLDRAM) si M_u dépasse cette taille.

5.5 L'impact sur les performances du traitement des paquets

Dans cette section, nous étudions l'impact des opérations de mise à jour des compteurs et de collecte des statistiques sur les performances du chemin de traitement des paquets du nœud de communication.

5.5.1 La compétition sur la mémoire des compteurs

FlowME exploite la structure des requêtes et des entrées de flots pour garder l'utilisation des ressources du nœud de communication à un minimum. Il est légitime de poser la question sur l'impact de FlowME sur les performances du nœud de communication. D'une part, parce que la mise à jour des compteurs fait partie du traitement des paquets, et de l'autre côté parce qu'il y a deux processus qui sont en compétition sur la mémoire des compteurs : la routine de mise à jour des compteurs au niveau du NPU et le processus de lecture des valeurs des compteurs du *host*.

Les deux opérations de mise à jour et de lecture des compteurs par le NPU et le *host* sont des opérations atomiques. Rappelons que pour une opération atomique, le processeur envoie la commande de l'opération et continue son traitement, et ceci sans attendre la confirmation que la commande s'est exécutée. Le NPU envoie des commandes d'incrémentation de compteur et continue son traitement de paquet sans attendre la confirmation. Le *host* utilise le même type d'opération pou lire et initialiser les compteurs. Les opérations sur la mémoire des compteurs n'ont pas d'impact sur les performances de traitement paquets, puisqu'ils ne causent pas de latence pour traitement des paquets.

5.5.2 L'impact sur le temps de traitement du paquet

Notre implémentation exploite le processus de classification pour les opérations de traitement des paquets du nœud de communication. Le temps total de traitement des paquets est composé du temps de classification, d'incrémentation du compteur de flot et d'application des actions sur le paquet. Dans notre banc d'essai, le temps moyen des opérations de classification et d'application des actions sur le paquet est estimé à 2447 ns. Le processeur réseau utilisé à une

cadence de 400 MHz. Avec notre stratégie qui concentre la surveillance uniquement sur les flots couverts par les requêtes des utilisateurs, les paquets des flots restants sont automatiquement accélérés.

Nous avons mis en place une routine dans la phase de résolution de paquets (*resolving*) qui incrémente les compteurs de flots en fonction de la référence récupérée de la phase de classification du paquet. Pour mesurer le délai de traitement supplémentaire ajouté par la routine d'incrémentation de compteur de flots, nous plaçons deux estampilles temporelles (*timestamps*), avant et après la routine d'incrémentation. Un étalonnage est effectué pour éliminer le temps dû au placement des estampilles. Nous avons mesuré un temps de 9 cycles d'horloge nécessaire pour incrémenter un compteur. Ce temps inclus l'extraction de la référence du compteur à incrémenter et l'envoi de la commande d'incrémentation vers l'ordonnanceur de la mémoire des compteurs (statistics block). En tenant compte de la cadence de l'horloge, FLOWME rajoute un délai de 22.5 ns au traitement du paquet.

5.6 Conclusion

Nous avons évalué dans ce chapitre notre stratégie de surveillance de trafic qui se base sur les treillis. Les métriques que nous avons utilisées sont le temps d'établissement de la structure de mesure, le temps de mise à jour du treillis par l'ajout d'entrées de flots et le délai de traitement de compteur de flot. Les résultats expérimentaux ont montré une réduction considérable du nombre de compteurs matériels utilisés, avec un effort global de calcul raisonnable et un impact négligeable sur le traitement des paquets.

CONCLUSION

Dans ce travail, nous avons proposé FLOWME, une solution de surveillance de trafic réseau reposant sur les treillis. L'objectif du travail est d'améliorer les performances de surveillance du trafic basé sur les flots de paquets et donc de minimiser le nombre des compteurs matériels utilisés.

Un constat important que nous avons fait est que les solutions actuelles de surveillance de trafic ignorent largement les associations qui existent entre la sémantique des requêtes et les descripteurs des flots traités par le nœud de communication.

Dans notre approche, nous avons considéré comme cruciales ces associations, c'est pourquoi nous avons utilisé une structure de treillis qui nous a permis d'exprimer correctement ces associations, tout en ayant la richesse et la flexibilité pour soutenir la recherche d'une affectation optimale des compteurs.

Un point fort du travail est de pouvoir exploiter les associations qui existent entre la sémantique des requêtes et les descripteurs des flots traités par le nœud de communication pour permettre une affectation de compteurs mieux informée et plus ciblée. Les principaux avantages de l'approche de surveillance de trafic reposant sur les treillis sont l'efficacité dans le calcul des statistiques et l'utilisation optimale des ressources. En outre, notre méthode est facile à mettre en œuvre tout en étant très flexible et adaptable à un large éventail de contextes d'utilisation.

Les résultats de validation de FLOWME sur les distributions de trafic, de requêtes et d'entrées de flots de nos expériences montrent une réduction jusqu'à 90% sur le coût mémoire, et moins de $30ms$ en moyenne pour mettre à jour la structure de treillis par ajout d'un flot.

Notre approche ne nécessite pas un traitement particulier des paquets ou l'ajout de structures de recherche supplémentaires. Le nœud de communication doit faire appel juste à une instruction qui incrémente un compteur associé au flot du paquet. L'impact sur les performances du chemin de traitement des paquets du nœud de communication est donc presque inexistant.

Un point faible de notre approche est la préparation de la structure de treillis de base. En effet, FLOWME calcule le treillis d'associations des flots aux champs d'entête, avant d'introduire les requêtes utilisateurs. Ce prétraitement des flots représente un coût supplémentaire.

Une manière de lever la limitation du prétraitement est de construire le contexte d'entrée avec les flots comme objets et les requêtes comme attributs. Ensuite, utiliser les sous-hiérarchies de Galois pour représenter directement les associations entre les flots traités par le nœud de communication et les requêtes des utilisateurs.

À l'issue de ce travail, nous avons montré que les treillis fournissent un moyen simple pour établir et parcourir les associations entre les requêtes utilisateurs et les flots traités par le nœud de communication. L'exploitation de leurs propriétés nous a permis de déployer les compteurs d'un nœud de communication d'une façon optimale.

La surveillance du trafic réseau est un domaine riche. Nous pouvons explorer par exemple dans un travail futur l'analyse de traces de trafic en ligne ou *a posteriori* pour découvrir les motifs fréquents.

La généricité de la solution mathématique rend l'approche particulièrement adaptée aux protocoles réseau émergents (CDN [4], NDN [16], etc.).

APPENDICE A

LA TRACE D'EXÉCUTION DE L'ALGORITHME DE CONSTRUCTION DU TREILLIS DE CONCEPTS

Ci-après, le traitement des trois premières itérations de l'algorithme 1 pour construire le treillis des concepts.

$C = \{(\{f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7\}, \varnothing)\}$

$c = (\{f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7\}, \varnothing)$

$Children = \varnothing$

$h = h_1$ $(h' = \{f_0, f_1, f_4, f_5\})$

$F_h = \{f_0, f_1, f_4, f_5\}$

$Children \leftarrow Children \cup (\{f_0, f_1, f_4, f_5\}, \{h_1\})$

$h = h_2$ $(h' = \{f_2, f_3\})$

$F_h = \{f_2, f_3\}$

$Children \leftarrow Children \cup (\{f_2, f_3\}, \{h_2\})$

$h = h_3$ $(h' = \{f_6, f_7\})$

$F_h = \{f_6, f_7\}$

$Children \leftarrow Children \cup (\{f_6, f_7\}, \{h_3\})$

$h = h_4$ $(h' = \{f_0, f_1\})$

$F_h = \{f_0, f_1\}$

$Children \leftarrow Children \cup (\{f_0, f_1\}, \{h_4\})$

$h = h_5$ $(h' = \{f_1, f_5\})$

$F_h = \{f_1, f_5\}$

$Children \leftarrow Children \cup (\{f_1, f_5\}, \{h_5\})$

$h = h_6$ $(h' = \{f_2, f_3, f_6, f_7\})$

$F_h = \{f_2, f_3, f_6, f_7\}$

$Children \leftarrow Children \cup (\{f_2, f_3, f_6, f_7\}, \{h_6\})$

$h = h_7$ $(h' = \{f_0, f_1, f_4, f_5\})$

$F_h = \{f_0, f_1, f_4, f_5\}$

F_h *existe déjà dans* $(\{f_0, f_1, f_4, f_5\}, \{h_1\})$, *ajouter* $\{h_7\}$ *à son intent:* $(\{f_0, f_1, f_4, f_5\}, \{h_1, h_7\})$

$h = h_8$ $(h' = \{f_2, f_3, f_7\})$

$F_h = \{f_2, f_3, f_7\}$

$Children \leftarrow Children \cup (\{f_2, f_3, f_7\}, \{h_8\})$

$h = h_9$ $(h' = \{f_0, f_4, f_6\})$

$F_h = \{f_0, f_4, f_6\}$

$Children \leftarrow Children \cup (\{f_0, f_4, f_6\}, \{h_9\})$

$h = h_{10}$ $(h' = \{f_1, f_3, f_5, f_6, f_7\})$

$F_h = \{f_1, f_3, f_5, f_6, f_7\}$

$Children \leftarrow Children \cup (\{f_1, f_3, f_5, f_6, f_7\}, \{h_{10}\})$

connecter c avec les concepts maximaux : $max(Children) = \{(\{f_2, f_3, f_6, f_7\}, \{h_6\}),$

$(\{f_0, f_1, f_4, f_5\}, \{h_1, h_7\}), (\{f_0, f_4, f_6\}, \{h_9\}), (\{f_1, f_3, f_5, f_6, f_7\}, \{h_{10}\})\}$

$c = (\{f_1, f_3, f_5, f_6, f_7\}, \{h_{10}\})$

$Children = \varnothing$

$h = h_1$ $(h' = \{f_0, f_1, f_4, f_5\})$

$F_h = \{f_1, f_5\}$

$Children \leftarrow Children \cup (\{f_1, f_5\}, \{h_1, h_{10}\})$

$h = h_2$ $(h' = \{f_2, f_3\})$

$F_h = \{f_3\}$

$Children \leftarrow Children \cup (\{f_3\}, \{h_2, h_{10}\})$

$h = h_3$ $(h' = \{f_6, f_7\})$

$F_h = \{f_6, f_7\}$

$Children \leftarrow Children \cup (\{f_6, f_7\}, \{h_3, h_{10}\})$

$h = h_4$ $(h' = \{f_0, f_1\})$

$F_h = \{f_1\}$

$Children \leftarrow Children \cup (\{f_1\}, \{h_4, h_{10}\})$

$h = h_5$ $(h' = \{f_1, f_5\})$

$F_h = \{f_1, f_5\}$

F_h *existe déjà dans* $(\{f_1, f_5\}, \{h_1\})$, *ajouter* $\{h_5\}$ *à son intent:* $(\{f_1, f_5\}, \{h_1, h_{10}, h_5\})$

$h = h_6$ $(h' = \{f_2, f_3, f_6, f_7\})$

$F_h = \{f_3, f_6, f_7\}$

$Children \leftarrow Children \cup (\{f_3, f_6, f_7\}, \{h_6, h_{10}\})$

$h = h_7$ $(h' = \{f_0, f_1, f_4, f_5\})$

$F_h = \{f_1, f_5\}$

F_h *existe déjà dans* $(\{f_1, f_5\}, \{h_1, h_5\})$, *ajouter* $\{h_7\}$ *à son intent:* $(\{f_1, f_5\}, \{h_1, h_{10}, h_5, h_7\})$

$h = h_8$ $(h' = \{f_2, f_3, f_7\})$

$F_h = \{f_3, f_7\}$

$Children \leftarrow Children \cup (\{f_3, f_7\}, \{h_8, h_{10}\})$

$h = h_9$ $(h' = \{f_0, f_4, f_6\})$

$F_h = \{f_6\}$

$Children \leftarrow Children \cup (\{f_6\}, \{h_9, h_{10}\})$

connecter c avec les concepts maximaux : $max(Children) = \{(\{f_3, f_6, f_7\}, \{h_6, h_{10}\}),$

$(\{f_1, f_5\}, \{h_1, h_{10}, h_5, h_7\})\}$

$c = (\{f_2, f_3, f_6, f_7\}, \{h_6\})$

$Children = \varnothing$

$h = h_1$ $(h' = \{f_0, f_1, f_4, f_5\})$
$F_h = \{\varnothing\}$
$Children \leftarrow Children \cup (\varnothing, \{h_6, h_1\})$
$h = h_2$ $(h' = \{f_2, f_3\})$
$F_h = \{f_2, f_3\}$
$Children \leftarrow Children \cup (\{f_2, f_3\}, \{h_6, h_2\})$
$h = h_3$ $(h' = \{f_6, f_7\})$
$F_h = \{f_6, f_7\}$
$Children \leftarrow Children \cup (\{f_6, f_7\}, \{h_6, h_3\})$
$h = h_4$ $(h' = \{f_0, f_1\})$
$F_h = \{\varnothing\}$
F_h *existe déjà dans* $(\varnothing, \{h_6, h_1\})$, *ajouter* $\{h_4\}$ *à son extent :* $(\varnothing, \{h_6, h_1, h_4\})$
$h = h_5$ $(h' = \{f_1, f_5\})$
$F_h = \{\varnothing\}$
F_h *existe déjà dans* $(\varnothing, \{h_6, h_1, h_4\})$, *ajouter* $\{h_5\}$ *à son extent :* $(\varnothing, \{h_6, h_1, h_4, h_5\})$
$h = h_7$ $(h' = \{f_0, f_1, f_4, f_5\})$
$F_h = \{\varnothing\}$
F_h *existe déjà dans* $(\varnothing, \{h_6, h_1, h_4, h_5\})$, *ajouter* $\{h_7\}$ *à son extent :* $(\varnothing, \{h_6, h_1, h_4, h_5, h_7\})$
$h = h_8$ $(h' = \{f_2, f_3, f_7\})$
$F_h = \{f_2, f_3, f_7\}$
$Children \leftarrow Children \cup (\{f_2, f_3, f_7\}, \{h_6, h_8\})$
$h = h_9$ $(h' = \{f_0, f_4, f_6\})$
$F_h = \{f_6\}$
$Children \leftarrow Children \cup (\{f_6\}, \{h_6, h_9\})$
$h = h_{10}$ $(h' = \{f_1, f_3, f_5, f_6, f_7\})$
$F_h = \{f_3, f_6, f_7\}$
$Children \leftarrow Children \cup (\{f_3, f_6, f_7\}, \{h_6, h_{10}\})$
connecter c avec les concepts maximaux : $max(Children) = \{(\{f_2, f_3, f_7\}, \{h_6, h_8\}),$
$(\{f_3, f_6, f_7\}, \{h_6, h_{10}\})$

APPENDICE B

LA THÉORIE DE TREILLIS ET OPTIMALITÉ DES ALGORITHMES

La caractérisation formelle du demi-treillis des projections est un travail d'équipe et est présenté dans l'article [29].

B.1 La caractérisation formelle du demi treillis des projections

Le demi-treillis des projections contient l'ensemble des concepts cibles, projections et bases. Dans cette section nous étudions les propriétés de chacun afin de pouvoir les identifier rapidement.

B.1.1 La caractérisation formelle des concepts-cibles

L'ensemble des requêtes est lié naturellement au treillis des flots du fait que chaque requête peut être affectée à un *concept cible* unique, qui est identifié par l'ensemble des flots qui le satisfont, $\{q\}'$:

Définition B.1. *Pour toute requête $q \subseteq \mathcal{H}$, son concept cible du contexte $\mathcal{K}(\mathcal{F}, \mathcal{H}, \mathcal{M})$ est donné par la fonction : $\gamma : \wp(\mathcal{H}) \rightarrow \mathcal{C}_{\mathcal{K}}$ avec $\gamma(q) = (\{q\}', \{q\}'')$*

Par sa définition même, $\gamma(q)$ satisfait $E(\gamma(q))$ qui est l'ensemble de réponses pour q, c'est-à-dire, les entrées de flots requis pour le calcul de statistiques. Dans des termes plus généraux, quelle que soit la requête $q \subseteq \mathcal{H}$, son ensemble réponse est l'extension d'un concept dans le treillis. Pour fonder formellement notre raisonnement sur les statistiques sur les requêtes, nous établissons d'abord une structure de support pour Q dans les $\mathcal{L}_{\mathcal{K}}$. Les concepts cibles sont à la base de celle-ci.

Définition B.2. *Étant donné $Q \subseteq \wp(\mathcal{H})$, son* ***ensemble de concepts cibles*** *dans $\mathcal{K}(\mathcal{F}, \mathcal{H}, \mathcal{M})$*

est $T(Q) = \{\gamma(q) \mid q \in Q\}$.

B.1.2 La caractérisation formelle des concepts-projection

Considérons la sous-structure générée par $T(Q)$ et ses intersections: d'après le théorème fondamental de l'AFC, elle comprend toutes les bornes inférieures des concepts cibles (sauf l'ensemble vide). Le surensemble résultant de $T(Q)$ est appelé l'ensemble des *concepts projection*:

Définition B.3. *Étant donné un ensemble de requêtes* $Q \subseteq \wp(\mathcal{H})$*, l'ensemble de ses concepts* ***projection*** *du contexte* $\mathcal{K}(\mathcal{F}, \mathcal{H}, M)$ *est* $P(Q) = \{\bigwedge X \mid X \subseteq T(Q);\ X \neq \varnothing\}$.

En outre, $P(Q)$ forme clairement un inf-demi-treillis par rapport à l'ordre $\leqslant_{\mathcal{K}}$, notée $\mathcal{L}_{|Q} = \langle P(Q), \leqslant_{\mathcal{K}|P(Q)}\rangle$, le *demi-treillis de projections*. Tester si un concept est une projection est délicat, car le calcul de toutes les bornes inférieures des concepts cibles pourrait être coûteux. Nous attribuons donc à chaque concept $c = (F, H)$ un vecteur de bits auxiliaire pour tenir compte des requêtes satisfaites par les flots de son extension:

Définition B.4. *Étant donné un concept* $c = (F, H)$*, son vecteur-requêtes* $v(c)$ *est une chaîne de bits qui repère les* q_i *correspondant à* H*:*

$$v((F,H))[i] = \begin{cases} 1, & si\ q_i \subseteq H \\ 0, & sinon \end{cases} \qquad 1 \leqslant i \leqslant N$$

Le tableau B.1 donne les vecteur-requêtes des concepts du treillis.

Vecteur-requêtes	Concepts	Vecteur-requêtes	Concepts
00000	$\mathbf{c_4}, c_{11}, c_{12}, c_{15}$	00111	$c_0, \mathbf{c_2}$
10000	$\mathbf{c_6}, c_{13}, c_{14}, c_{16}, c_{17}, c_{18}$	10101	$\mathbf{c_{10}}$
01000	$\mathbf{c_5}$	11000	$\mathbf{c_7}$
10111	$\mathbf{c_3}$	00101	$\mathbf{c_8}, c_9$
11111	$\mathbf{c_1}$		

Tableau B.1: Les valeurs des vecteurs-requêtes pour les concepts du treillis

(les projections sont en gras)

Le treillis contient toutes les intersections des extensions des concepts cibles. Les intersections maximales des concepts cibles sont les extensions des concepts projection. Nous identifions

donc un concept c comme projection s'il est exactement la borne inférieure des concepts cibles des requêtes dans $v(c)$.

Propriété B.5. $c \in P$ *ssi* $c = \bigwedge\{\gamma(q_i) \mid v(c)[i] = 1\}$.

Le concept c est la borne inférieure d'un ensemble de concepts cibles, donc la valeur de $v(c)$ est maximale par rapport au vecteurs requêtes de ces parents dénotés c^U. Le concept c est reconnu donc du moment qu'il rajoute une requête que ses concepts parents ne possèdent pas : $c \in P$ ssi $\forall \bar{c} \in c^U, v(\bar{c}) \neq v(c)$. Pour vérifier qu'un concept est projection, l'algorithme peut vérifier que $|v(c)| > \max_{\bar{c} \in c^U}(|v(\bar{c})|)$. Par exemple, $v(c_{10})$ a trois 1, plus que ses parents, c_8 (un) et c_6 (deux), le concept c_{10} passe le test de projection (en effet, $c_{10} = c_6 \wedge c_8$).

B.1.3 La caractérisation formelle des concepts-bases

Pour réaliser l'association entre les flots et les requêtes, considérons l'association des flots aux concepts projection: pour un flot f couvert avec moins d'une $q \in Q$, il y a *un concept minimal unique* dans $P(Q)$ qui le détient. Nous appelons ce concept, *le concept-base* de f. Formellement, nous définissons d'abord un mappage de concepts:

Définition B.6. *Le mappage* $\mu_Q : \mathcal{F} \to C_K$ *associe chaque* f *à la borne inférieure de toutes les projections qui la détiennent, on l'appel son* ***concept base****:*

$$\mu_Q(f) = \bigwedge\{(F, H) \mid (F, H) \in P(Q);\ f \in F\}.$$

Par définition de $P(Q)$, pour tout f couvert par certaines q, $\mu_Q(f)$ est une projection de $P(Q)$ qui est en dessous de toute autre projection qui détient f.

Propriété B.7. *Étant donné un contexte* $\mathcal{K}(\mathcal{F}, \mathcal{H}, \mathcal{M})$*, un ensemble de requêtes* $Q \subseteq \wp(\mathcal{H})$ *et un flot* $f \in \mathcal{F}$ *tel qu'au moins un* q *couvre* f*, le concept* $\mu_Q(f)$ *est le concept-projection minimal contenant* f*.*

L'ensemble des projections bases $G(Q)$ (ou simplement G) est défini par:

Définition B.8. *Étant donné un ensemble de requêtes* $Q \subseteq \wp(\mathcal{H})$*, leurs* ***concepts-bases*** *dans le contexte* $\mathcal{K}(\mathcal{F}, \mathcal{H}, M)$ *sont :* $G(Q) = \{\mu_Q(f) \mid f \in \mathcal{F}\} - \top$,

Pour résumer, nous avons trouvé une famille de concepts dans le treillis qui définit une partition de l'ensemble global des flots $\mathcal{F}$.

B.2 La caractérisation des concepts d'un treillis actualisé

Ici, nous étudions l'évolution du treillis des concepts $\mathcal{L}_{\mathcal{K}}$ au treillis $\mathcal{L}_{\mathcal{K}_n}$ par ajout d'un nouveau flot f_n. Expliquons la similarité structurelle mathématiquement: le treillis $\mathcal{L}_{\mathcal{K}}$ est isomorphe à un sous-ordre de $\mathcal{L}_{\mathcal{K}_n}$. Comme justification de ce fait, observons la façon dont les familles respectives des intensions se rapportent. En fait, comme $\mathcal{H}$ reste stable, toutes les intensions de $\mathcal{K}$ demeurent des intensions dans $\mathcal{K}_n$.

Propriété B.9. *Étant donné $\mathcal{K}$ et f_n, il s'ensuit que $C^h_{\mathcal{K}} \subseteq C^h_{\mathcal{K}_n}$.*

Par exemple, dans la figure 2.2, les intensions des concepts c_0, c_5, c_8, c_9, et c_{12} sont exactement les mêmes que dans le treillis initial (comme le sont les intensions de tous les concepts qui n'ont pas été étendus, par exemple, c_6 et c_7). La stratégie globale que nous allons suivre revient à obtenir $C^h_{\mathcal{K}_n}$ de $C^h_{\mathcal{K}}$ et $\{f_n\}'$: Les *intensions* de $C^h_{\mathcal{K}_n}$ manquantes dans $C^h_{\mathcal{K}}$ représentent les intersections $\{f_n\}' \cap H_o$ pour certains $H_o \in C^h_{\mathcal{K}}$.

Propriété B.10. *Dans $\mathcal{K}_n$, nous avons $C^h_{\mathcal{K}_n} = C^h_{\mathcal{K}} \cup \{\{f_n\}' \cap H_o \mid H_o \in C^h_{\mathcal{K}}\}$.*

En conséquence, les *extensions* des concepts dans $C_{\mathcal{K}_n}$ ont l'une des deux seules formes possibles: F ou $F \cup \{f_a\}$ où $F \in C^f_{\mathcal{K}}$.

Propriété B.11. *$F \in C^f_{\mathcal{K}_n}$ implique que $F - \{f_n\} \in C^f_{\mathcal{K}}$.*

Par exemple, l'extension de c_{20}, $\{f_0, f_4, f_8\}$ est l'extension de c_9 (voir la figure 2.1), augmentée de f_8 . En revanche, les extensions de $C^f_{\mathcal{K}}$ peuvent apparaître telles qu'elles sont dans $C^f_{\mathcal{K}_n}$ (propriété B.12 ligne 1) ou générer un équivalent étendu avec f_n (propriété B.12 ligne 2). Les deux cas étant non exclusives.

Propriété B.12. *Dans $\mathcal{K}_n$, pour chaque $(F_o, H_o) \in C_{\mathcal{K}}$:*

$$F_o \in C^f_{\mathcal{K}_n} \text{ ssi } H_o \nsubseteq f_n'$$
$$(F_o \cup \{f_n\}) \in C^f_{\mathcal{K}_n} \text{ ssi } H_o = (H_o \cap f_n')''$$

Maintenant, nous allons montrer comment les ensembles T, P et G peuvent être correctement transformés en T_n, P_n et C_n, respectivement, avec un effort de calcul minimal.

Premièrement, observons que pour $c \in C_{\mathcal{K}}$, $v(c)$ conserve sa valeur dans $C_{\mathcal{K}_n}$:

Propriété B.13. *Pour un concept $c = (F, H) \in C_{\mathcal{K}_n}$, si $H \in C^h_{\mathcal{K}}$ alors $v_n(c) = v(\bar{c})$ tel que $\bar{c} = (H', H)$.*

La raison est que $v(c)$ ne dépend que de $\mathcal{H}$ et Q qui restent stables dans $\mathcal{K}_n$. Ainsi, la fonction $v_n()$ évolue à partir de $v()$ simplement en calculant les valeurs des nouveaux concepts

dans C_n.

Maintenant, T_n peut se différencier de T lorsque certains $q \in Q$ changent de cibles ($\gamma(q) \neq \gamma_n(q)$). Il est clair que la nouvelle cible de q, $\gamma_n(q)$ est un nouveau concept de $\mathcal{K}_n$ seulement si $\gamma(q)$ est son géniteur dans $\mathcal{K}$:

Propriété B.14. *Pour une requête $q \in Q$ tel que $\gamma(q) \neq \gamma_n(q)$ (reciblée), $\bar{c} = \gamma_n(q)$ est un nouveau concept dans $\mathcal{C}_{\mathcal{K}_n}$ tel que $c = \gamma(q)$ est son géniteur.*

Cela découle de la minimalité de H'' (fermeture de H) par rapport aux intensions comprenant H. Ainsi, pour $c = (F, H)$ et $c_n = (F_n, H_n)$, montrons que $H = H''_n$ (d'où le statut de géniteur) dans $\mathcal{K}$. En effet, en supposant que $H \neq H''_n$, alors $H''_n \subset H$ (*) et puisque $q \subseteq H$ (rappelons que $q'' = H$) et H_n est la fermeture de q dans $\mathcal{K}_n$ (l'intension minimale comprenant q). Pourtant, étant donné que $q \subseteq H_n \subset H''_n$, (*) serait contradictoire avec la minimalé de $H = q''$ dans $\mathcal{K}$.

P_n émerge à partir de P selon deux scénarios distincts: (1) comme pour T_n, un nouveau concept peut devenir projection en éclipsant son géniteur dans P_n, et (2) un nouveau concept peut devenir la borne inférieure d'un ensemble de cibles de requêtes qui n'a pas d'équivalent dans P. Rappelons que les projections sont identifiés au sein de P par $v()$ (Propriété B.5). En d'autres termes, dans le premier cas, l'infimum c d'un ensemble de cibles ($v(c)$) se déplace vers un concept différent $\dot{c}$ qui a la même valeur de vecteur requête ($v(c) = v_n(\dot{c})$), alors que dans le deuxième cas, il émerge un ensemble de cibles $v_n(\dot{c})$ qui n'existait pas précédemment.

Propriété B.15. *Étant donné un $c \in P_n$, si c n'est pas l'équivalent du concept de projection $\bar{c} = \bigwedge\{\gamma(q_i) \mid v_n(c)[i] = 1\}$ dans $\mathcal{K}$, alors c est un nouveau concept avec $\bar{c}$ comme son géniteur.*

Supposons que pour certains $c = (F, H) \in P_n$, la projection $\bar{c} = \bigwedge\{\gamma(q_i) | v_n(c)[i] = 1\}$ de P est tel que $\bar{c} = (F_o, H_o)$ et $H_o \neq H$ ($\bar{c}$ n'est pas un concept équivalent dans $\mathcal{K}$, même si $v(c) = v(\bar{c})$). Cela signifie que $F \neq F_o$, et puisqu'ils sont l'intersection des extensions de cibles à partir de $v(c)$, il s'ensuit que *toutes ces extensions* ont changé. Comme nous l'avons vu précédemment, la seule évolution possible de l'extension d'une cible pour une requête q dans $\mathcal{K}_n$ est d'augmenter de f_n. Par conséquent, leur intersection F (corollaire de la propriété B.5) comprend f_n aussi, et puisque H n'est pas une intension de $\mathcal{K}$, c est un nouveau concept. Par le même argument, F_o ne peut qu'être $F_o = F - \{f_n\}$, donc $\bar{c}$ est le géniteur de c.

Dans le deuxième cas, la nouvelle projection c est aussi un nouveau concept:

Propriété B.16. *Étant donné un $c = (F, H) \in P_n$, si pour toutes les projections $\bar{c} \in P$,*

$v_n(c) \neq v(\bar{c})$, *alors c est un nouveau concept.*

En supposant le contraire, soit $H \in \mathcal{C}^h_{\mathcal{K}}$, donc c n'est pas un nouveau concept ($f_n \notin F$) et donc $v_n(c) = v(c)$. Par conséquent, il s'agit d'un concept avec la même valeur de vecteur requête dans $\mathcal{K}$, c'est c lui-même, ce qui signifie qu'il doit y avoir un concept maximal $\bar{c}$, tel que $v(c) = v(\bar{c})$, c'est-à-dire, une borne inférieure. Ceci contredit l'hypothèse de départ.

G_n étant un sous-ensemble de P_n, il se maintient similairement: Dans le premier cas ci-dessus, tous les flots qui sont basés au géniteur - qui disparaît à partir de P_n, donc à partir de G_n - doivent être ré-basés dans la nouvelle projection c_n. Dans le deuxième cas, aucun des flots de $\mathcal{F}$ ne peut être basé sur c_n, étant donné que leurs concepts de flots dans $\mathcal{C}_{\mathcal{K}_n}$ ont des intensions dans $\mathcal{C}^h_{\mathcal{K}}$. Ainsi, les vecteurs requêtes respectives ne changent pas dans $\mathcal{K}_n$, donc tous les flots pareils sont basés sur c pour lequel $v_n(c)$ existe dans P. Par conséquent, f_n est le seul candidat pour les bases du deuxième.

Pour résumer, dans T_n, les nouveaux concepts des concepts géniteurs cibles saisissent les requêtes ciblées appartenant à leurs intensions respectives, tandis que les géniteurs sans requêtes restantes disparaissent de T_n. Pour P_n, les nouveaux concepts sont testés pour la projection et, si elle c'est positive, les géniteurs le sont aussi. En G_n, les flots basés sur une projection mouvante se deplacent du géniteur au nouveau concept. Finalement, $\mu(f_n)$ est trouvé.

B.3 Optimalité des algorithmes

B.3.1 Le coût de construction du treillis et complexité algorithmique

La construction du treillis est une tâche coûteuse puisque les treillis sont des structures complexes dont la taille peut atteindre jusqu'à une fonction exponentielle des dimensions du contexte, c'est-à-dire $\mid \mathcal{F} \mid$ or $\mid \mathcal{H} \mid$. Les algorithmes de construction du treillis adoptent des procédés de listage combinatoire puisqu'ils énumèrent toutes les solutions du problème sous-jacent. Il existe cependant d'autres façons d'aborder la tâche, la plus directe consiste à lister toutes les bicliques maximales d'un graphe bipartite. Une présentation détaillée des différentes approches de la construction du treillis peut être trouvée dans [1].

En ce qui concerne le coût de calcul, bien que la taille d'un treillis de concept pourrait croître exponentiellement, dans des situations concrètes, elle reste polynomiale, notamment avec des ensembles de données dispersées. Avec ces données, chaque individu possède seulement une

petite fraction des attributs (voir, par exemple, Chapitre 2 dans [1]). En outre, en raison des fortes variations de tailles des treillis, il est notoirement difficile d'estimer leur taille avec des paramètres généraux. Ainsi l'évaluation de la complexité des algorithmes de construction du treillis implique généralement la taille de la sortie, c'est-à-dire $\mid \mathcal{C}_\mathcal{K} \mid$, en tant que paramètre. La fonction minimale connue du coût du temps d'un algorithme de construction du treillis est $O(lm)$ où $l = \mid \mathcal{C}_\mathcal{K} \mid$ et $m = \mid \mathcal{M} \mid$.

Plusieurs algorithmes ont été conçus pour maintenir un treillis de concepts en ligne, dit incrémental. Leur complexité théorique est généralement comparable à celle des algorithmes de traitement par lots (patch), soit $O(lm)$ (voir par exemple [28]).

B.3.2 L'exactitude et minimalité de l'affectation des compteurs

Dans ce qui suit, nous allons montrer que notre affectation des compteurs basée sur $T(Q)/G(Q)$ est: (1) exacte, et (2) de cardinalité minimale.

Maintenant, *l'exactitude* signifie que pour chaque q, l'ensemble total des flots rapportant les valeurs de leurs compteurs à la somme de q est $E(\gamma(q))$. Soit $S(q)$ l'ensemble des flots correspondant aux compteurs rapportés à la somme de q. Par définition, $S(q) = \{f \mid \mu_Q(f) \leqslant_\mathcal{K} \gamma(q)\}$. Le théorème B.17 montre que les flots sont bien rapportés dans l'ensemble $S(q)$ du moment qu'ils correspondent à la requête q.

Théorème B.17. $\forall q \in Q, f \in \mathcal{F}$, $f \in S(q)$ *ssi* $q \subseteq f'$.

La minimalité signifie qu'aucune affectation de compteurs à partir d'un ensemble plus restreint de compteurs ne pourrait répondre également à toutes les q de Q. Nous nous concentrons sur la partition sous-jacente de $\mathcal{F}$. Le théorème suivant, montre qu'une partition de taille inférieure strictement à $|G|$ ne permet pas de répondre à certaines requêtes.

Théorème B.18. *Soit* $cpt : \mathcal{F} \rightarrow \wp(\mathcal{F})$ *avec* $cpt(f) = F$ *ssi* $f \in F$, *et supposons que la cardinalité de l'image* $|cpt(\mathcal{F})| < |G|$. *Alors* $\exists q \in Q$ *tel que l'ensemble réponse* $E(\gamma(q))$ *n'est pas décomposable en une union des ensembles de* $cpt(\mathcal{F})$.

En résumé, nous avons fourni un moyen de structurer l'association des requêtes aux flots au sein du treillis. Ces associations font émerger des sous-ensembles auxquels les compteurs sont liés et qui forment une partition optimale.

Théorème B.19. $\forall q \in Q, f \in \mathcal{F}$, $f \in S(q)$ *ssi* $q \subseteq f'$.

Pour montrer que $q \subseteq f'$ ssi $\mu_Q(f) \leqslant \gamma(q)$, ce qui équivaut à $f \in E(\gamma(q))$ ssi $\mu_Q(f) \leqslant \gamma(q)$, il suffit d'observer que $\mu_Q(f)$ est minimal entre les projections ayant f et que $\gamma(q)$ est l'une de ces projections.

Théorème B.20. *Soit* $cpt : \mathcal{F} \to \wp(\mathcal{F})$ *avec* $cpt(f) = F$ *ssi* $f \in F$, *et supposons que la cardinalité de l'image* $|cpt(\mathcal{F})| < |G|$. *Alors* $\exists q \in Q$ *tel que l'ensemble réponse* $E(\gamma(q))$ *n'est pas décomposable en une union des ensembles de* $cpt(\mathcal{F})$.

En raisonnant par l'absurde, supposons que toutes les $E(\gamma(q))$ représentent des unions de $cpt(f)$ pour des flots f d'un ensemble bien choisi. Un raisonnement combinatoire direct emmène à $\exists f_1, f_2 \in \mathcal{F}$ qui n'ont pas le même concept-base, $\mu_Q(f_1) \neq \mu_Q(f_2)$ alors qu'ils existent dans le même sous-ensemble de $cpt(\mathcal{F})$, $cpt(f_1) = cpt(f_2)$. Pourtant, l'existence dans le même sous-ensemble de $cpt(\mathcal{F})$ signifie qu'ils répondent à des ensembles de requêtes différents: $\{q \mid q \subseteq f_1'\} \neq \{q \mid q \subseteq f_2'\}$, donc sans perte de généralité on peut supposer $\exists q_a \in Q$, tel que $q_a \subseteq f_1'$ mais $q_a \nsubseteq f_2'$. Cependant, cela conduit à une contradiction, car il n'existe aucun moyen de se décomposer correctement $E(\gamma(q_a))$ en une union d'ensembles de $cpt(\mathcal{F})$ (rappelons que $cpt(f_1) = cpt(f_2)$): $cpt(f_1)$ est nécessairement à l'intérieur, mais il n'y a aucun moyen de supprimer la contribution de f_2 (soustraction indisponible).

APPENDICE C

LES TREILLIS DE CONCEPTS

C.1 Le treillis de concepts initial FHM

Tableau C.1: Extensions et intensions des concepts du treillis initial FHM

Concept	Extension	Intension
c_0	f_0	h_1, h_4, h_7, h_9
c_1	$\varnothing$	$h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, h_{10}$
c_2	f_0, f_1	h_1, h_4, h_7
c_3	f_1	$h_1, h_4, h_5, h_7, h_{10}$
c_4	$f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7$	$\varnothing$
c_5	f_2, f_3	h_2, h_6, h_8
c_6	f_1, f_3, f_5, f_6, f_7	h_{10}
c_7	f_3	h_2, h_6, h_8, h_{10}
c_8	f_0, f_1, f_4, f_5	h_1, h_7
c_9	f_0, f_4	h_1, h_7, h_9
c_{10}	f_1, f_5	h_1, h_5, h_7, h_{10}
c_{11}	f_2, f_3, f_6, f_7	h_6
c_{12}	f_0, f_4, f_6	h_9
c_{13}	f_3, f_6, f_7	h_6, h_{10}
c_{14}	f_6	h_3, h_6, h_9, h_{10}
c_{15}	f_2, f_3, f_7	h_6, h_8
c_{16}	f_6, f_7	h_3, h_6, h_{10}
c_{17}	f_3, f_7	h_6, h_8, h_{10}
c_{18}	f_7	h_3, h_6, h_8, h_{10}

C.2 Le treillis de concepts étendu eFHM

Tableau C.2: Extensions et intensions des concepts du treillis étendu eFHM

Concept	Extension	Intension
c_0	f_0	h_1, h_4, h_7, h_9
c_1	$\varnothing$	$h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, h_{10}$
c_2	f_0, f_1	h_1, h_4, h_7
c_3	f_1	$h_1, h_4, h_5, h_7, h_{10}$
c_4	$f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8$	$\varnothing$
c_5	f_2, f_3	h_2, h_6, h_8
c_6	f_1, f_3, f_5, f_6, f_7	h_{10}
c_7	f_3	h_2, h_6, h_8, h_{10}
c_8	f_0, f_1, f_4, f_5	h_1, h_7
c_9	f_0, f_4	h_1, h_7, h_9
c_{10}	f_1, f_5	h_1, h_5, h_7, h_{10}
c_{11}	f_2, f_3, f_6, f_7	h_6
c_{12}	f_0, f_4, f_6, f_8	h_9
c_{13}	f_3, f_6, f_7	h_6, h_{10}
c_{14}	f_6	h_3, h_6, h_9, h_{10}
c_{15}	f_2, f_3, f_7	h_6, h_8
c_{16}	f_6, f_7	h_3, h_6, h_{10}
c_{17}	f_3, f_7	h_6, h_8, h_{10}
c_{18}	f_7	h_3, h_6, h_8, h_{10}
c_{19}	f_0, f_1, f_4, f_5, f_8	h_7
c_{20}	f_0, f_4, f_8	h_7, h_9
c_{21}	f_2, f_3, f_8	h_2
c_{22}	f_8	h_2, h_7, h_9

BIBLIOGRAPHIE

[1] Claudio Carpineto et Giovanni Romano : *Concept data analysis: Theory and applications.* Wiley, 2004.

[2] H Jonathan Chao et Bin Liu : *High performance switches and routers.* John Wiley & Sons, 2007.

[3] Cisco Systems : Netflow white paper. http://www.cisco.com/en/US/products/ps6601/prod_white_papers_list.html, 2012.

[4] John Dilley, Bruce Maggs, Jay Parikh, Harald Prokop, Ramesh Sitaraman et Bill Weihl : Globally distributed content delivery. *Internet Computing, IEEE*, 6(5):50–58, 2002.

[5] Cristian Estan, Ken Keys, David Moore et George Varghese : Building a better netflow. *SIGCOMM Comput. Commun. Rev.*, 34(4):245–256, août 2004.

[6] Cristian Estan, Stefan Savage et George Varghese : Automatically inferring patterns of resource consumption in network traffic. *In Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 137–148. ACM, 2003.

[7] Cristian Estan et George Varghese : New directions in traffic measurement and accounting. *SIGCOMM Comput. Commun. Rev.*, 32(4):323–336, août 2002.

[8] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford et Fred True : Deriving traffic demands for operational ip networks: Methodology and experience. *IEEE/ACM Transactions on Networking (ToN)*, 9(3):265–280, 2001.

[9] OE Ferkouss, Omar Mounaouar, Ilyas Snaiki, Hamza Dahmouni, R Ben Ali, Yves Lemieux et Omar Cherkaoui : A 100gig network processor platform for openflow. *In Proceeding of the 7th conference on Network and Service Management (CNSM)*, pages 1–4. IEEE, 2011.

[10] T. Ganegedara, Weirong Jiang et V. Prasanna : Frug: A benchmark for packet forwarding in future networks. *In Proc. of the 29th IEEE Intl. Perform. Comp. & Comm. Conf. (IPCCC'10),*, pages 231 –238, dec. 2010.

[11] B. Ganter et R. Wille : *Formal Concept Analysis, Mathematical Foundations.* Springer-Verlag, 1999.

[12] Bernhard Ganter, Gerd Stumme et Rudolf Wille : *Formal Concept Analysis: foundations and applications*, volume 3626. springer, 2005.

[13] Farzad Ghannadian, Li Fang et Michael J. Quinn : Adaptive, flow-based network traffic measurement and monitoring system, U.S. Patent 7 639 613, Dec 2009.

[14] Robert Godin, Rokia Missaoui et Hassan Alaoui : Incremental concept formation algorithms based on galois (concept) lattices. *Computational Intelligence*, 11(2):246–267, 1995.

[15] Pankaj Gupta et Nick McKeown : Algorithms for packet classification. *Network, IEEE*, 15(2):24–32, 2001.

[16] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs et Rebecca L. Braynard : Networking named content. *In Proceeding of the 5th Intl. Conf. on Emerging networking experiments and technologies*, CoNEXT'09, pages 1–12, New York, NY, USA, 2009. ACM.

[17] Lavanya Jose, Minlan Yu et Jennifer Rexford : Online measurement of large traffic aggregates on commodity switches. *In Proc. of the USENIX HotICE workshop*, 2011.

[18] Faisal Khan, Lihua Yuan, Chen-Nee Chuah et Soheil Ghiasi : A programmable architecture for scalable and real-time network traffic measurements. *In Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, pages 109–118, New York, NY, USA, 2008. ACM.

[19] Open Networking Foundation : Openflow switch specification version 1.3, Jun 2012.

[20] Équipe Orpailleur Loria : The Coron System. http://coron.loria.fr, mai 2012.

[21] A. Ramachandran, S. Seetharaman, N. Feamster et V. Vazirani : Fast monitoring of traffic subpopulations. *In Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 257–270. ACM, 2008.

[22] Frederic Raspall : Efficient packet sampling for accurate traffic measurements. *Computer Networks*, 56(6):1667–1684, 2012.

[23] Devavrat Shah, Sundar Iyer, B Prahhakar et Nick McKeown : Maintaining statistics counters in router line cards. *Micro, IEEE*, 22(1):76–81, 2002.

[24] Cisco Systems : Cisco nexus 7000 hardware architecture. http://www.valleytalk.org/wp-content/uploads/2013/04/BRKARC-3470.pdf, janvier 2007.

[25] David E Taylor : Survey and taxonomy of packet classification techniques. *ACM Computing Surveys (CSUR)*, 37(3):238–275, 2005.

[26] EZchip Technologies : Ezchip np-4 product brief. http://www.ezchip.com/Images/pdf/NP-4_Short_Brief_online.pdf, avril 2011.

[27] P. Valtchev, R. Missaoui et R. Godin : *Formal Concept Analysis for Knowledge Discovery and Data Mining: The New Challenges.* LNCS v.2961. Springer, 2004.

[28] Petko Valtchev, Rokia Missaoui et Robert Godin : A framework for incremental generation of closed itemsets. *Discrete Appl. Math.*, 156:924–949, March 2008.

[29] Petko Valtchev, Omar Mounaouar, Omar Cherkaoui, Alexandar Dimitrov et Laurent Marchand : FlowME: Lattice-based Traffic Measurement. *arXiv*, 2012.

[30] Lihua Yuan, Chen-Nee Chuah et Prasant Mohapatra : *ProgME: towards programmable network measurement*, volume 19. IEEE Press, Piscataway, NJ, USA, février 2011.