AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Analyse de trafic HTTPS pour la supervision d'activités utilisateurs

THÈSE

présentée et soutenue publiquement le 14 décembre 2020

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Pierre-Olivier Brissaud

Composition du jury

Rapporteurs :	Radu State	Professeur, Université du Luxembourg
	Sandrine Vaton	Professeur, IMT Atlantique
Examinateurs :	Gregory Blanc	Maître de conférences, TELECOM SudParis
	Jean-Noël Colin	Professeur, Université de Namur
	Marine Minier	Professeur, Université de Lorraine (Loria)
Invités :	Olivier Bettan	Responsable Laboratoire R&D Cybersécurité, Thales SIX GTS France (Encadrant Industriel)
Encadrants :	Isabelle Chrisment	Professeur, Université de Lorraine (Loria) (Directrice de thèse)
	Jérôme François	Chargé de recherche, Inria Grand Est (Co-directeur de thèse)

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

Cette thèse est le fruit d'un long travail dont l'aboutissement est beaucoup dû à l'aide et au soutien de quelques personnes que je tiens à remercier.

Je tiens tout d'abord à remercier les rapporteurs et les examinateurs de cette thèse pour l'intérêt et le temps qu'ils ont consacré à mes travaux de recherche malgré le contexte difficile. Je les remercie d'avoir accepté de faire partie de mon jury de thèse.

Je remercie très sincèrement Isabelle Chrisment (directrice de thèse) et Jérôme François (co-directeur de thèse) pour leur soutien et leur encadrement durant l'ensemble de ma thèse. J'ai énormément apprécié travailler avec eux, que ce soit d'un point de vue scientifique ou humain. De plus, je les remercie pour leur accueil au sein de l'équipe RESIST.

Je tiens également à remercier Olivier Bettan (encadrant industriel) pour son accueil au sein de l'équipe cyber à Palaiseau. Merci pour ton aide et ta convivialité.

Mes remerciements s'adressent aussi à l'ensemble de mes collègues de l'équipe RESIST au loria ainsi qu'à ceux de l'équipe Cyber à Thales Palaiseau. J'ai toujours pu évoluer dans une ambiance de travail bénéfique malgré ma double localisation. Merci à mes collègues et/ou amis pour ces échanges professionnels ou personnels et leur bonne humeur.

J'adresse un remerciement particulier à plusieurs personnes avec lesquelles j'ai pu avoir la chance de travailler, dont la précieuse collaboration a permis de me guider ou à l'obtention de résultats. Un grand merci à Thibault Cholez, Charles Xosanavongsa, Romain Ferrari, Tarek Marce et Evrard Callot.

Pour finir, je remercie ma famille et en premier lieu mes parents qui m'ont toujours accompagné durant l'ensemble de mes études. Que ce soit par le temps consacré, des moyens ou encore des encouragements. Je n'oublie pas ma fiancée, Soline, je la remercie pour son aide, sa patience et sa gentillesse.

A toutes et à tous merci

A mes parents.
Mes grands-parents.

Sommaire

Introduction générale

Contexte 1
Problématique 5
Contributions 6
Organisation de la thèse 7

Partie I État de l'Art **9**

Chapitre 1 Le protocole HTTPS

1.1 Introduction 12
1.2 Sécurité des échanges sur Internet : les protocoles SSL/TLS 12
1.2.1 L'évolution de SSL 2.0 à TLS 1.3 13
1.2.2 Le protocole TLS 1.2 13
1.3 HTTP et son évolution 17
1.3.1 HTTP/0.9 : version initiale 17
1.3.2 HTTP/1 : extension du protocole 17
1.3.3 HTTP/1.1 : standardisation 18
1.3.4 HTTP/2 : simplification et performances 18
1.3.5 HTTP/3 : une révision en profondeur 19
1.4 HTTP/1.1 et HTTP/2 : détail et comparaison 19
1.4.1 Etablissement de la connexion 19
1.4.2 Les modèles d'échanges 21
1.4.3 Le format des messages 23
1.5 Synthèse 27

Chapitre 2 Les méthodes d'analyse de trafic chiffré

2.1 Introduction 30
2.2 Détection de protocoles 30
2.2.1 Trafic TLS 31
2.2.2 Trafic HTTPS 32
2.3 Détection de services 32
2.3.1 Utilisation d'informations en clair 33
2.3.2 Website fingerprinting 34
2.3.3 Mobile service classification 36
2.4 Détection d'actions utilisateur sur un service HTTPS 36
2.4.1 Solutions actives 36
2.4.2 Solutions passives 37
2.5 Synthèse 38

Partie II Analyse de trafic chiffré HTTP/1.1 41

Chapitre 3 Modélisation et méthode pour l'analyse du trafic chiffré HTTP/1.1

3.1 Introduction 44
3.2 Modélisation du trafic chiffré HTTP/1.1 44
3.2.1 Modélisation initiale 44
3.2.2 Caractérisation d'une trace par des images 45
3.2.3 Modèle raffiné grâce à la détection de la taille des images chiffrées . . 47
3.3 Méthode 48
3.3.1 Vue d'ensemble de notre solution 49
3.3.2 Extraction des caractéristiques 50
3.3.3 Génération des signatures 52
3.3.4 Algorithme de classification 57
3.4 Synthèse 59

Chapitre 4 Évaluation de la classification : H1Classifier

4.1 Introduction . . . 62
4.2 Présentation du jeu de données . . . 62
4.2.1 Génération des traces . . . 62
4.2.2 Jeu de données . . . 63
4.3 Expérimentation et résultats . . . 64
4.3.1 Métriques pour l'évaluation . . . 65
4.3.2 Évaluation des paramètres . . . 67
4.3.3 Expérience à grande échelle . . . 69
4.3.4 Discussion . . . 72
4.4 Classifier du trafic HTTP/2 avec H1Classifier . . . 73
4.4.1 Recherche d'une nouvelle caractéristique . . . 74
4.4.2 Évaluation de H1Classifier avec du trafic HTTP/2 . . . 75
4.4.3 Limitations du portage de la méthode . . . 76
4.4.4 Test du caractère discriminant des signatures . . . 77
4.5 Synthèse . . . 79

Partie III Analyse de trafic chiffré HTTP/2 **81**

Chapitre 5 Modélisation et méthode pour l'analyse du trafic chiffré HTTP/2

5.1 Introduction . . . 84
5.2 Modélisation du trafic chiffré HTTP/2 . . . 84
5.2.1 Modélisation initiale . . . 84
5.2.2 Présentation des caractéristiques . . . 85
5.2.3 Modélisation raffinée . . . 87
5.3 Méthode de classification pour le trafic chiffré HTTP/2 . . . 87
5.3.1 Vue d'ensemble . . . 88
5.3.2 Extraction des caractéristiques à partir d'un flux réseau . . . 89
5.3.3 Classification par apprentissage supervisé . . . 90
5.3.4 Forêt d'arbres décisionnels . . . 91
5.4 Synthèse . . . 94

Chapitre 6 Évaluation de H2Classifier

6.1 Introduction 98
6.2 H2Classifier : paramétrage et évaluation 98
6.2.1 Jeux de données 99
6.2.2 Optimisation des paramètres 101
6.2.3 Évaluation de la qualité de la classification 107
6.3 Impact temporel 113
6.3.1 Jeu de données 113
6.3.2 Performance de H2Classifier dans la durée 114
6.3.3 Apprentissage régulier du modèle 115
6.4 Portabilité de H2Classifier à d'autres services web 118
6.4.1 Jeu de données 118
6.4.2 Expérience et analyse 119
6.5 Impact de l'environnement de capture 121
6.5.1 Jeu de données 121
6.5.2 Expérience et analyse 122
6.6 Synthèse 124

Conclusion générale

Travail réalisé 127
Perspectives 129

Productions **131**

Glossaire **133**

Table des figures **137**

Liste des tableaux **139**

Annexe

Réglages des paramètres H2Classifer (figures) 141

Bibliographie **149**

Résumé **157**

Abstract **158**

Introduction générale

Contexte

Plusieurs scandales liés à la surveillance de masse ou à des fuites de données ont attiré l'attention du public sur le sujet de la protection des communications. On peut par exemple citer les révélations par Edward Snowden sur le projet PRISM [1] de surveillance généralisée aux États-Unis de la NSA, ou Golden Shield [2] pour son pendant chinois. Ce manque de confiance dans la sécurité des communications suscite un besoin de protection des données. Les utilisateurs sont maintenant plus vigilants et commencent à privilégier dans leurs choix de services ceux proposant plus de sécurité.

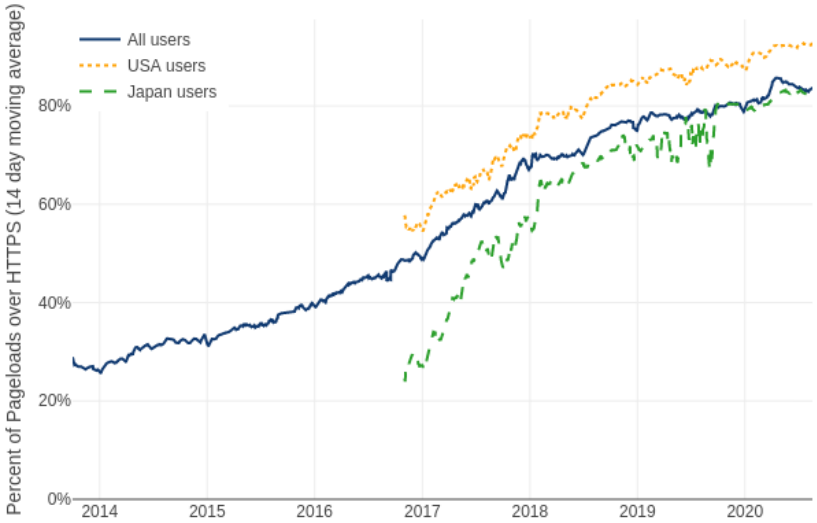
La porte d'entrée sur Internet pour le grand public se fait le plus souvent via l'usage d'un navigateur. Ainsi, la protection du protocole HTTP, qui permet le transport des données entre un serveur et un navigateur web, est une priorité au vu de son usage central dans le quotidien des utilisateurs du web. HTTPS, la version sécurisée du protocole HTTP, est l'agrégation du protocole applicatif HTTP et du protocole de chiffrement TLS. HTTPS voit donc son usage se généraliser pour faire face à ce besoin de sécurité. Par exemple, on constate que, depuis février 2017, plus de la moitié des pages web visitées dans le monde sont déjà sécurisées à l'aide du protocole HTTPS, comme en atteste l'article de Electronic Frontier Foundation [3]. Cet article fait référence à la courbe présentée dans la figure 1(a) qui mesure le pourcentage de pages chargées sur le navigateur Firefox en fonction du temps. Ces résultats sont confirmés par une autre courbe issue du rapport de Google pour le navigateur Chrome, tracé dans la figure 1(b). Outre le passage de la barre des 50%, ces figures montrent une augmentation continue au cours du temps. Ainsi, entre début 2017 et aujourd'hui (mars 2020), où plus de 81% des pages sont maintenant protégées par HTTPS, le protocole est devenu central dans l'usage quotidien des utilisateurs du web.

L'adoption du protocole HTTPS peut également être attribuée à l'évolution des outils pour faciliter son déploiement. Ainsi, fin 2015 est lancée l'autorité de certification Let's Encrypt qui donne pour la première fois accès à des certificats gratuits et reconnus par les navigateurs comme certificats de confiance. La même année, la RFC pour HTTP/2 est validée [1]. Elle définit une version chiffrée (h2) et une autre non chiffrée (h2c). Cependant, dans les faits, les navigateurs classiques tels que Firefox, Chrome ou Safari ne supportent pas h2c afin de forcer la version sécurisée [4]. De fait, ces différents éléments ont contribué massivement à la généralisation de l'usage du chiffrement pour le protocole HTTP (HTTPS).

Cependant, la généralisation du chiffrement rend les tâches de supervision plus complexes

1. https://en.wikipedia.org/wiki/PRISM_(surveillance_program)
2. https://en.wikipedia.org/wiki/Golden_Shield_Project
3. https://www.eff.org/deeplinks/2017/02/were-halfway-encrypting-entire-web
4. https://wiki.mozilla.org/Networking/http2

(a) Firefox
(`https://letsencrypt.org/stats/`)



(b) Chrome
(`https://transparencyreport.google.com/https/overview`)

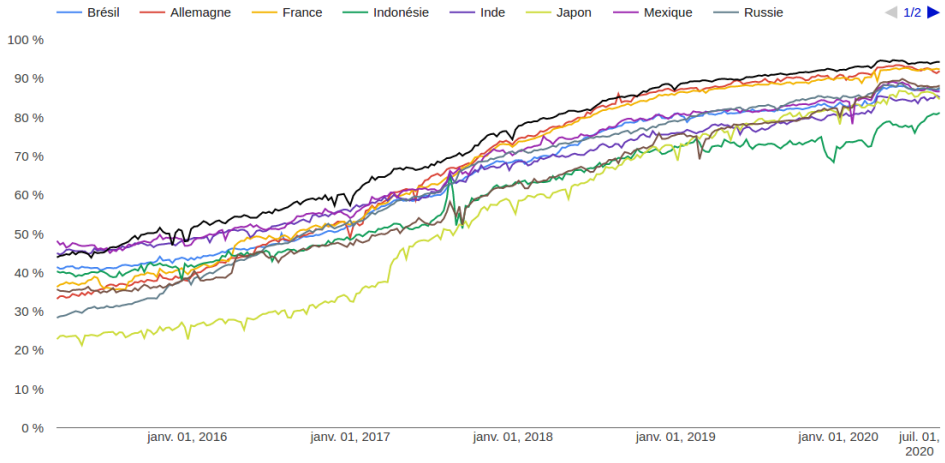


FIGURE 1 – Pourcentage des pages HTTPS ouvertes avec un navigateur en fonction du temps

car les outils disponibles auparavant pour du trafic en clair ne sont plus adéquats. Deux besoins antagonistes sont ainsi apparus. D'un côté, il est important de garantir la confidentialité des échanges pour protéger les communications et par la même occasion améliorer la protection de la vie privée des utilisateurs de l'Internet. D'un autre côté, les actions de détection et de surveillance face à des internautes aux comportements illicites ou malveillants doivent pouvoir, pour des raisons d'ordre légal, être opérées malgré la présence du chiffrement. En effet, de part son étendue, l'Internet à de plus en plus d'impacts sur le monde réel. Un usage malveillant peut avoir des répercussions via des perturbations ou des dommages physiques bien réels. Par exemple d'après un rapport d'Accenture [2], le coût d'une attaque cyber est évalué en moyenne à 13 millions de dollars (2018). La figure 2 montre les conséquences de ces attaques qui entraînent des pertes de données, des baisses du chiffre d'affaire et même des dégâts matériels. Notre objectif est donc de répondre en même temps à ces deux besoins, a priori contradictoires. Il faut ainsi permettre la détection de certaines actions spécifiques tout en gardant la protection assurée par le chiffrement.

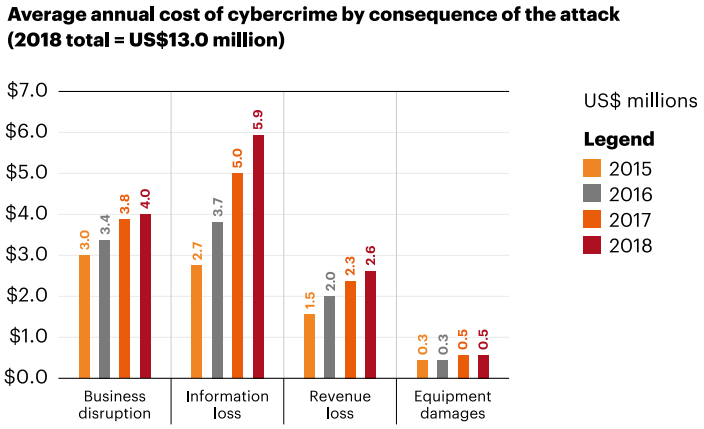


FIGURE 2 – Coût annuel moyen par type de conséquences d'une attaque cyber (extrait de [2])

A l'évidence, la solution classique d'analyse de trafic, telle que l'inspection des paquets, n'est plus adaptée car elle nécessite que le contenu soit disponible en clair. Pour appliquer cette méthode dans le cadre d'un trafic chiffré, il faut donc faire usage de proxy de déchiffrement, et rendre le trafic non chiffré. Cela portant directement atteinte à la vie privée des utilisateurs, nous avons exclu cette option de nos travaux. Les autres solutions classiques comme le filtrage des ports TCP ou des adresses IP sont aussi impactées par le chiffrement ou la généralisation des hébergements mutualisés dans le cloud. Il est ainsi nécessaire de rechercher d'autres méthodes.

Malgré la présence du chiffrement, il est tout de même possible d'observer la « forme » des échanges (nombre de paquets, tailles, direction, etc...). Nous nous sommes ainsi intéressés aux méthodes d'analyse de trafic passives, qui reposent sur la construction et la reconnaissance de signatures caractéristiques pour les comportements que l'on souhaite superviser, via l'observation de la « forme » du trafic. Il existe un large pan de la littérature qui utilise ces approches pour reconnaître par exemple l'utilisation d'un protocole, l'installation d'un logiciel ou encore l'accès à une page web. On distingue ainsi trois niveaux de détection : le niveau protocole, le niveau service et le niveau action utilisateur. Pour le protocole HTTPS, les différents niveaux de détection sont les suivants :

— Protocole : détection de l'usage du protocole HTTPS.

— Service : détection du service demandé (nom de domaine, exemple : google.com).

— Action utilisateur : reconnaissance de l'usage d'un utilisateur au sein d'un service.

Le dernier niveau de détection, correspondant aux actions utilisateur, est le plus complexe à atteindre sans l'accès au contenu des requêtes, et c'est donc sur sujet que nos travaux s'orientent. Lorsqu'un utilisateur veut accéder à du contenu sur le web, la solution la plus courante consiste à effectuer une recherche au moyen d'un mot-clé sur un service. C'est pourquoi nous proposons, dans cette thèse, d'étudier comment détecter la recherche de mots-clés correspondant à un usage non légitime d'un utilisateur au sein d'un service.

Cette approche est innovante par rapport à l'état de l'art car elle permet de détecter des comportements spécifiques, définis à l'avance, lors de l'utilisation d'un service par un utilisateur. La plupart des travaux sur l'analyse de trafic passive décrits dans la littérature s'intéressent à la détection de services ou de protocoles, mais rarement au comportement d'un utilisateur. Nous sommes les seuls, à notre connaissance, à détecter la requête d'un mot-clé sur un service si ce mot-clé fait partie d'une liste prédéfinie de termes, dans du trafic HTTPS. Détecter les usages au sein d'un service, apporte une solution intermédiaire aux administrateurs, entre laisser le plein accès à ce service ou le bannir à cause de quelques comportements illicites.

La détection de mauvais comportements n'est pas synonyme de surveillance globale. Notre solution n'a pas vocation à identifier le comportement complet de chaque utilisateur. Le but est de lever des alertes ou de bloquer le trafic d'un utilisateur qui ne respecterait pas la loi ou des règles préétablies, dans le cadre d'un réseau d'entreprise par exemple. De ce fait, nous proposons dans ce travail une solution qui permet de détecter ce type de comportement en définissant des règles a priori. Utiliser cet ensemble de règles (pouvant être mises à jour), permet de ne révéler que les requêtes et les réponses qui enfreignent ces règles, et d'ainsi préserver la confidentialité des communications et donc la vie privée des utilisateurs légitimes tout en maintenant une supervision du trafic pour les comportements non légitimes. Nos méthodes ne peuvent pas, en effet, détecter l'ensemble des comportements possibles.

HTTP est un protocole défini en 1991, qui a évolué pour être toujours plus performant et fiable. Il existe donc plusieurs versions de ce protocole (HTTP/0.9, HTTP/1, HTTP/1.1, HTTP/2 ou encore HTTP/3). Nos travaux s'intéressent à deux versions majeures (actuelles) du protocole HTTP qui cohabitent encore aujourd'hui : HTTP/1.1 et HTTP/2.

Malgré la publication de la RFC 7540 [1] dédiée à HTTP/2 en mai 2015, ce dernier, bien que standardisé, prend du temps à être adopté. En effet, en début d'année 2017 (début de nos travaux), seulement 11,2% des sites web supportaient HTTP/2 d'après le rapport w3techs [5]. De plus, le protocole HTTP/1.1 est, encore aujourd'hui, majoritairement utilisé sur l'ensemble des sites web, bien que la plupart des principaux services aient migré. Par exemple, sur le top 100 des sites web de Alexa [6], nous avons relevé 33 services qui ne supportaient pas HTTP/2, ou encore, fin mars 2020 w3techs relevait que seulement 44,2% des sites web supportaient HTTP/2. C'est pour ces raisons que l'étude de HTTP/1.1 reste importante.

D'un autre point de vue, d'après Keycdn, le 13 avril 2016 [7], 68% des requêtes diffusées par ce RDC (réseau de distribution de contenus) utilisaient le protocole HTTP/2. De plus, si on regarde la courbe d'adoption du protocole HTTP/2 en nombre de services, on observe une progression d'environ 10% par an depuis son lancement en 2015, comme on peut le voir dans le graphique

5. `https://w3techs.com/technologies/details/ce-http2`
6. `https://www.alexa.com/topsites`
7. `https://www.keycdn.com/blog/http2-statistics`

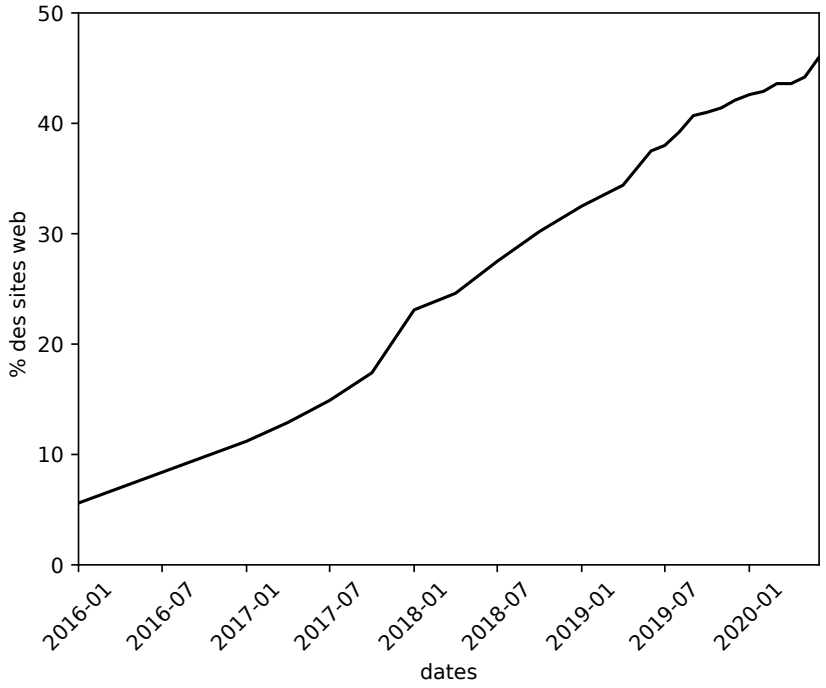


FIGURE 3 – Évolution de l'adoption de HTTP/2
(données provenant du w3techs)

de la figure 3 issue de w3techs[8]. Ainsi, bien que HTTP/2 ne soit pas majoritaire en nombre de services, il est actuellement majoritaire en termes de part de trafic. Pour ces raisons, il est intéressant pour nos travaux de considérer les versions 1.1 et 2 du protocole HTTP.

Problématique

Nous souhaitons définir une méthode capable de reconnaître un comportement utilisateur et, en particulier, reconnaître la recherche d'un mot-clé non légitime sur un service. Cette méthode devra supporter les contraintes suivantes :

— Passive : l'analyse s'effectue uniquement via l'écoute du trafic qui transite sur le réseau.

— Transparente : aucune installation n'est faite sur les terminaux ou les applications des utilisateurs.

— Respect de la vie privée : une telle solution ne doit rien divulguer sur une utilisation légitime d'un service et donc ne doit pas être généralisable à un ensemble non borné de comportements.

Concrètement, lors de la capture d'une trace réseau correspondant au chargement du résultat de la recherche d'un mot-clé, nous collectons une séquence de paquets chiffrés. Nous souhaitons savoir si le mot-clé à l'origine de cette séquence est légitime. S'il ne l'est pas, nous voulons l'identifier et déterminer le mot-clé recherché.

Comme illustré par la figure 4, notre problématique est de définir une méthode de classification qui permet de répondre à ces besoins.

8. `https://w3techs.com/diagram/history_technology/ce-http2`

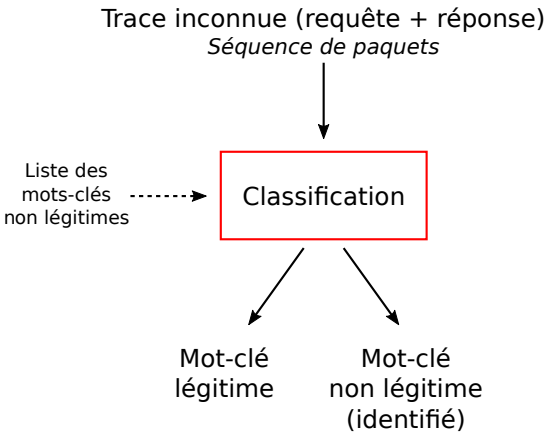


FIGURE 4 – Schématisation de la problématique

Cet objectif établi, nos recherches et les conditions opérationnelles rencontrées nous ont conduit à définir les contraintes et limitations suivantes :

— Nous faisons l'hypothèse que nous sommes capables de connaître le service auquel est associée chaque trace réseau que nous analysons. Cette hypothèse s'appuie sur le foisonnement de travaux performants dans ce premier niveau de classification.

— Nous ne nous intéressons pas aux réponses à adopter lors de la détection d'événements non légitimes. Cette étape est à la charge des utilisateurs (administrateurs) de la solution.

— Enfin, nous ne proposons pas de liste de mots-clés non légitimes à superviser. Cette tâche est également à la charge des administrateurs qui mettront en œuvre une telle solution. Nous précisons que, durant nos évaluations, les listes de mots-clés « non légitimes » sont donc purement arbitraire.

Contributions

Deux contributions principales sont présentées dans cette thèse. Elles définissent de nouvelles solutions de supervision pour détecter la recherche de mots-clés non légitimes pour un service sécurisé avec HTTPS. La première contribution étudie le trafic HTTPS lors de son utilisation avec la version 1.1 du protocole HTTP. La deuxième contribution traite, quant à elle, de la supervision du trafic HTTPS lorsqu'il est couplé avec le protocole HTTP/2.

Pour chacune de ces contributions, nous avons défini une solution théorique issue de la compréhension et de l'observation approfondie du trafic HTTP/1.1 ou HTTP/2. Ces solutions s'articulent autour de l'extraction de caractéristiques réseaux permettant d'identifier les différentes actions à superviser. Afin de combiner les informations apportées par les différentes caractéristiques réseaux, nous utilisons des techniques d'apprentissage automatique ainsi que des méthodes statistiques. Les deux solutions ont été implémentées et sont nommées dans la suite du document H1Classifier (classification du trafic HTTP/1.1) et H2Classifier (classification du trafic HTTP/2). Nous avons ensuite conduit pour chacune de ces implémentations des évaluations approfondies, pour lesquelles nous avons construit spécialement plusieurs larges ensembles de traces réseaux

que nous avons rendu disponibles publiquement [9]. Ces derniers sont constitués de traces réseau que nous avons capturées en considérant une large variété de configurations.

Organisation de la thèse

Nous avons découpé notre manuscrit en trois parties.

Partie 1 : État de l'art

La première partie introduit l'état de l'art autour du protocole HTTPS ainsi que l'analyse de trafic chiffré. Dans le premier chapitre nous commençons avec la présentation du protocole HTTPS, ou plus exactement des deux protocoles qui le composent : TLS et HTTP. TLS est un protocole de chiffrement de communication et HTTP un protocole applicatif pour le transfert des données. Dans ce chapitre, nous détaillons le fonctionnement de ces deux protocoles ainsi que les différentes versions existantes. En effet, que ce soit pour TLS ou HTTPS, il existe différentes versions du protocole, qui possèdent chacune leurs spécificités. Nous nous focalisons sur la version 1.2 du protocole TLS, et sur les versions 1.1 et 2 du protocole HTTP.

Dans le chapitre 2, nous présentons les méthodes d'analyse de trafic chiffré étudiées dans la littérature. Nous discutons les différents travaux suivant le niveau de détection auquel ils opèrent : niveau protocole, niveau service et enfin niveau action utilisateur.

Partie 2 : Analyse de trafic chiffré HTTP/1.1

Dans la deuxième partie, notre objectif est de reconnaître des mots-clés non légitimes lorsque nous observons du trafic chiffré HTTPS couplé avec HTTP/1.1. Les paquets que nous étudions étant chiffrés par le protocole TLS, nous ne pouvons donc récupérer que peu d'informations en observant leur contenu. Pour dépasser cette limitation, nous analysons les quantités d'informations échangées pour déterminer la nature des échanges (taille ou nombre des paquets par exemple...) . En pratique, malgré le chiffrement, nous sommes capables de récupérer la taille des objets HTTP chiffrés chargés au sein d'une page. Nous avons pu constater que pour des recherches distinctes sur un service, les pages retournées et donc les objets composant chaque page, sont spécifiques à la recherche qui en est à l'origine.

Nous modélisons ainsi le trafic comme étant une séquence de tailles d'objets HTTP chiffrés. À partir de ces informations, nous construisons une signature par page à l'aide d'une méthode statistique : l'estimation par noyau (KDE).

Dans le chapitre 3, nous présentons ensuite notre modélisation du trafic HTTPS (TLS + HTTP/1.1) ainsi que notre méthode de classification. Cette dernière nécessite une étape d'extraction de caractéristiques, la génération de signatures et enfin la création d'un algorithme de classification utilisant ces signatures. Ainsi dans le chapitre 4 nous présentons l'évaluation de notre méthode de classification (H1Classifier). Cette évaluation s'intéresse d'abord à la compréhension de l'impact de différents paramètres de H1Classifier, ainsi que ses performances dans un test à large échelle. Ensuite, nous analysons comment se comporte notre solution face à du trafic chiffré HTTP/2 et nous discutons finalement les limitations du portage de H1Classifier pour ce type de trafic.

9. `http://betternet.lhs.loria.fr/datasets/h2classifier/`

Partie 3 : Analyse de trafic chiffré HTTP/2

Dans la troisième partie de nos travaux, notre objectif est d'apporter une solution de classification capable de reconnaître l'usage de mots-clés non légitimes au sein d'un service pour du trafic HTTP/2. Suite aux résultats préliminaires peu convaincants sur l'usage de la solution H1Classifier pour du trafic HTTP/2, nous avons orienté nos travaux sur une nouvelle solution capable de supporter le trafic HTTP/2 (+ TLS). Nous avons repensé la méthode pour prendre en compte les contraintes remontées par les limitations de H1Classifier. L'utilisation de plusieurs caractéristiques issues du trafic nous a amenés à proposer une nouvelle solution faisant usage d'apprentissage automatique. Cette nouvelle méthode est décrite dans le chapitre 5.

Dans le chapitre 6, nous évaluons H2Classifier (l'implémentation de notre méthode) à travers différentes expériences. Nous étudions d'abord l'impact des différents paramètres, puis celui du nombre de mots-clés légitimes à superviser, ainsi que la proportion de traces issues de mots-clés légitimes par rapport aux non légitimes. Nous conduisons ensuite trois autres expériences permettant de mesurer les performances de H2Classifier dans le temps face à de nouveaux services et dans différents environnements. L'ensemble de ces travaux nous permettent d'évaluer la robustesse de notre solution et de vérifier sa viabilité pour une utilisation opérationnelle.

Première partie

État de l'Art

Chapitre 1

Le protocole HTTPS

Sommaire

1.1 Introduction 12
1.2 Sécurité des échanges sur Internet : les protocoles SSL/TLS . . 12
1.2.1 L'évolution de SSL 2.0 à TLS 1.3 13
1.2.2 Le protocole TLS 1.2 13
1.3 HTTP et son évolution 17
1.3.1 HTTP/0.9 : version initiale 17
1.3.2 HTTP/1 : extension du protocole 17
1.3.3 HTTP/1.1 : standardisation 18
1.3.4 HTTP/2 : simplification et performances 18
1.3.5 HTTP/3 : une révision en profondeur 19
1.4 HTTP/1.1 et HTTP/2 : détail et comparaison 19
1.4.1 Etablissement de la connexion 19
1.4.2 Les modèles d'échanges 21
1.4.3 Le format des messages 23
1.5 Synthèse 27

1.1 Introduction

Le protocole HTTPS ou "HyperText Transfer Protocol Secure" est utilisé dans le navigateur web et sécurise les communications entre ce dernier et les serveurs des sites contactés. D'une utilisation ciblée pour sécuriser la transmission de données sensibles comme des données bancaires, il est maintenant passé à un usage massif sur une grande quantité de services. Actuellement, d'après un rapport de Google [10] concernant les 100 sites web les plus visités dans le monde, ce qui représente environ 25% du trafic web mondial, la totalité supporte HTTPS et 96 sites utilisent ce protocole par défaut.

Le protocole HTTPS n'est pas un standard à proprement parler : il n'existe qu'une RFC informelle [3] à son sujet. En effet, HTTPS est uniquement défini comme la combinaison des protocoles HTTP et TLS, comme on utiliserait HTTP avec TCP dans le cas non sécurisé.

Dans la figure 1.1, nous présentons le positionnement des différents protocoles cités précédemment dans le modèle TCP/IP.

Notre objectif dans ce chapitre, est de présenter le protocole HTTPS pour comprendre son fonctionnement. Pour cela, nous commençons par décrire le protocole de chiffrement SSL/TLS. Ensuite nous détaillons le fonctionnement du protocole HTTP et en particulier pour les versions 1.1 et 2.

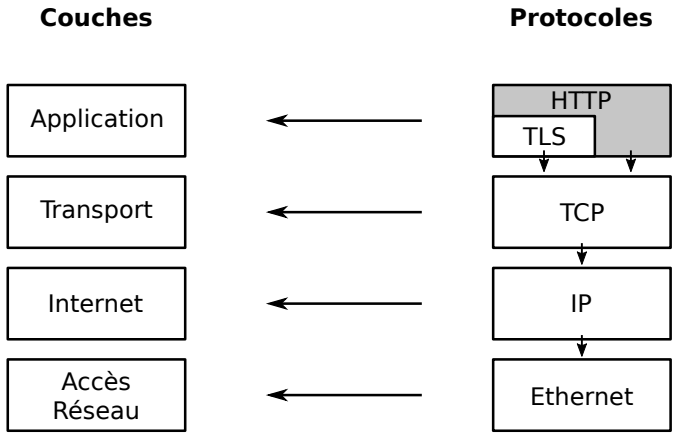


FIGURE 1.1 – Modèle TCP/IP et positionnement des protocoles

1.2 Sécurité des échanges sur Internet : les protocoles SSL/TLS

Avec le développement de l'Internet, le besoin de sécurité dans les communications s'est rapidement fait ressentir, en particulier avec l'essor du commerce en ligne. En effet, pour que les utilisateurs réalisent en toute confiance des achats en ligne, il fallait pouvoir garantir la sécurité du transfert des données. C'est dans ce contexte que le protocole SSL 2.0 a vu le jour en 1995. Son objectif était simple : protéger les informations sensibles qui transitent entre le client et le serveur.

Les différentes versions du protocole SSL, qui est devenu ensuite TLS, ont évolué en fonction des attaques dont elles ont été les cibles. Dans le premier paragraphe nous donnons un rapide aperçu de l'historique de l'évolution de SSL vers TLS. Nous précisons ensuite le fonctionnement

10. `https://transparencyreport.google.com/https/overview`

du protocole TLS dans sa version 1.2. Nous nous focalisons sur cette version que nous avons utilisée pour les travaux présentés dans ce manuscrit.

1.2.1 L'évolution de SSL 2.0 à TLS 1.3

L'évolution du protocole SSL/TLS couvre deux versions SSL, puis quatre versions TLS entre 1995 et aujourd'hui. SSL 2.0 est la première version du protocole SSL permettant de sécuriser les échanges sur Internet. Cette version ainsi que la suivante SSL 3.0 ont été développées par Netscape. En un an, plusieurs vulnérabilités ont été mises en lumière et a entraîné la sortie rapide de SSL 3.0 en 1996. Ce protocole a lui-même été remplacé en 1999 par TLS, qui repose sur SSL 3.0 mais a été défini, cette fois ci, par l'IETF et est légèrement plus sécurisé : il intègre par exemple le chiffrement AES et ne repose plus sur la fonction de hachage MD5, qui était déjà considérée comme faible du point de vue de la cryptanalyse. TLS 1.1 est une version transitoire qui a été adoptée en 2006. En 2008, l'IETF adopte les spécifications de TLS 1.2. Enfin, en 2018, la spécification de TLS 1.3 est standardisée.

Actuellement, les deux versions de SSL ont été respectivement bannies en 2011, avec la RFC 6176 [4], et en 2015, avec la RFC 7568 [5]. Le support des versions TLS 1.0 et 1.1 n'est plus d'actualité depuis le printemps 2020 dans les navigateurs modernes. En effet, Chrome [11], Firefox [12] et Edge [13] ont décidé d'arrêter le support de ces versions dans leur navigateur respectif. Il reste actuellement la version 1.2 qui est supportée depuis août 2008 et la dernière version, 1.3, qui s'installe progressivement. Le protocole TLS 1.3 [6] apporte trois principales nouveautés :

— La mise en place de la session TLS est limitée à un aller-retour au lieu de deux auparavant.

— La sécurité du protocole est renforcée grâce à l'abandon de certaines suites cryptographiques [14] qui ne supportent pas les standards de sécurité actuels.

— Les messages suivant le ServerHello durant le handshake sont à présent chiffrés.

Un récapitulatif des périodes d'activité des différents protocoles est disponible dans la figure 1.2.

1.2.2 Le protocole TLS 1.2

Le protocole TLS se place au dessus d'un protocole de transport fiable, très souvent TCP. TLS échange des messages sécurisés, appelés *records*. Afin de démarrer cet échange sécurisé, le client et le serveur communiquent un ensemble de messages initiaux, nommé handshake.

Nous décrivons ci-dessous cette étape de handshake, ainsi que la structure des records.

Mise en place d'une session TLS

Nous considérons le cas classique où le client est un navigateur web et le serveur est un service avec un certificat. Le handshake TLS pour la mise en place de la session TLS entre ces deux entités est composé d'un échange de plusieurs messages ou groupes de messages. La figure 1.3

11. `https://security.googleblog.com/2018/10/modernizing-transport-security.html`
12. `https://blog.mozilla.org/security/2018/10/15/removing-old-versions-of-tls/`
13. `https://blogs.windows.com/msedgedev/2018/10/15/modernizing-tls-edge-ie11/`
14. C'est la combinaison de quatre algorithmes pour paramétrer la sécurité d'une connexion. Cela comprend des algorithmes pour : l'échange des clés, l'authentification, le chiffrement symétrique et le hachage

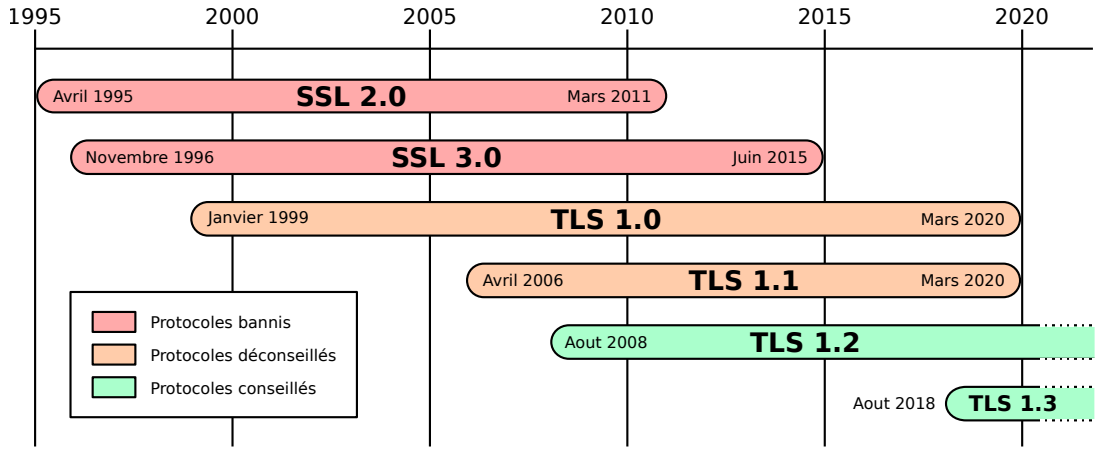


FIGURE 1.2 – Évolution de SSL/TLS dans le temps
(Extrait du rapport Enisa 2018 sur le chiffrement [7])

représente ces différentes séquences détaillées ci-après.

1. Suite à la mise en place de la connexion TCP, le client débute l'échange avec un message *ClientHello*. Ce message contient le numéro de la dernière version TLS supportée par le client, un nombre aléatoire, une liste de suites cryptographiques et une liste de formats de compression. Si le client et le serveur ont déjà une autre session active, un numéro de session (*session ID*) est envoyé. Cela permet de raccourcir le handshake dans le cas de l'actualisation d'une ancienne session. Dans ce message, le client peut également utiliser l'extension ALPN afin d'annoncer le protocole que TLS sécurisera. Le protocole ALPN est décrit dans le paragraphe 1.4.1.
2. A la réception du *ClientHello*, le serveur choisit la version du protocole TLS, la suite cryptographique et un format de compression à partir des possibilités énoncées par le client. L'ensemble des ces informations, ainsi qu'un nombre aléatoire, forment le message *ServerHello*. A la suite du message précédent, le serveur envoie un message *Certificate* qui contient son certificat afin que le client puisse l'authentifier. Avant d'envoyer le message *ServerHelloDone* qui indique la fin de la réponse du serveur, ce dernier peut être amené à envoyer un message *ServerKeyExchange*. Ce message contient des données nécessaires au client lors de l'utilisation de certaines suites cryptographiques.
3. Avec les différentes informations récupérées du serveur, le client génère une clé qu'il va chiffrer avec la clé publique du certificat du serveur. Cette clé est envoyée dans un message *ClientKeyExchange*. Avec cette clé et les nombres aléatoires du client et du serveur, ces deux derniers génèrent chacun de leur coté une même clé secrète. Cette dernière sécurisera le reste de la session. Après le message précédent, un message *ChangeCipherSpec* suivi d'un message chiffré *Finished* est envoyé. Ils servent respectivement à signifier la fin de la négociation et à tester si le déchiffrement s'est bien passé côté serveur.
4. Si le déchiffrement du message *Finished* s'est déroulé sans problème, le serveur envoie à son tour un message *ChangeCipherSpec* suivi d'un message chiffré *Finished*. Le client tente également de son coté de déchiffrer le message. Si cela fonctionne, le client peut envoyer des données applicatives en utilisant des records de type *Application Data*. La session est établie.

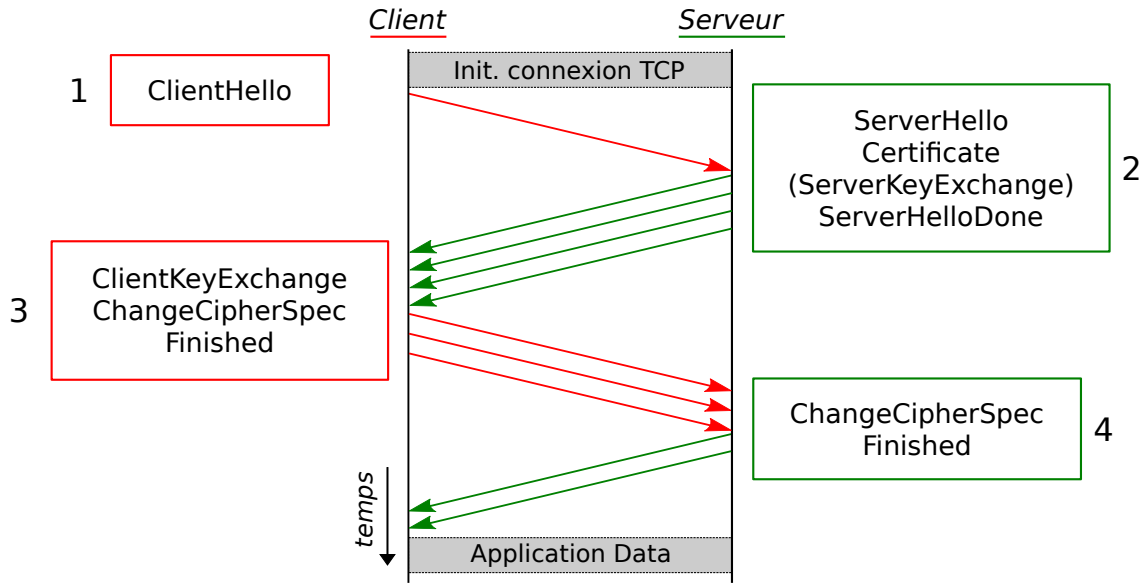


FIGURE 1.3 – Handshake TLS avec authentification du serveur

La couche Record de TLS

Tous les messages générés par le protocole TLS sont appelés records. Ces derniers ont une structure qui débute par une entête binaire, schématisée sur la figure 1.4. Dans les entêtes, on trouve dans l'ordre : le type du record, la version de TLS et la longueur du corps du message. Les différents types de records sont au nombre de quatre :

— change_cipher_spec (0x14) : ce type de records permet d'annoncer des changements au niveau du chiffrement.

— alert (0x15) : ces records apparaissent en cas de problème avec le protocole TLS du côté du client ou du serveur pour prévenir l'autre entité. Il existe deux niveaux d'alerte. Le premier est *warning* : la session peut continuer mais ne pourra pas être réutilisée. Le second est *fatal* : dans ce cas, la session est interrompue immédiatement.

— handshake (0x16) : ces records transportent les informations nécessaires à l'établissement de la session comme présenté dans le paragraphe précédent.

— application_data (0x17) : ce dernier type de record correspond au transport sécurisé des données de la couche applicative (par exemple HTTP).

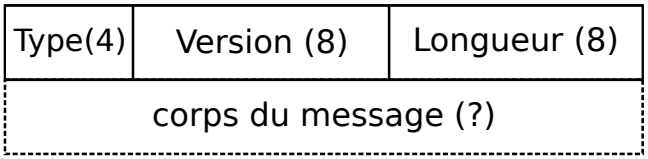


FIGURE 1.4 – Entête TLS (en bits)

Le corps du message des records qui ne sont pas du type *handshake* est authentifié et chiffré. Le protocole effectue différentes étapes afin de passer du contenu en clair au contenu sécurisé. Ces étapes sont au nombre de quatre, comme présenté dans la figure 1.5 avec :

1. Fragmentation : le message en clair est fragmenté pour produire des blocs de taille inférieure à 2^{14} octets.
2. Compression : les fragments peuvent être compressés en fonction du format défini lors de la négociation de la session. La taille après compression ne peut excéder $2^{14} + 1024$ octets. Cette marge est fixée afin de ne pas créer d'erreurs (et donc ralentir le processus) si la compression génère des fragments compressés de taille plus importante qu'avant compression (ce qui peut arriver dans le cas de la compression sans perte).

   Concernant la compression, il est important de noter qu'il est recommandé de ne pas l'activer. En effet, en septembre 2012, l'attaque CRIME (Compression Ratio Info-leak Made Easy) fut dévoilée par J. Rizzo et T. Duong [8]. Cette dernière permettait de récupérer des informations sur le contenu chiffré des records lorsque la compression était active.
3. Authentification : pour cette étape, le protocole calcule un HMAC (Hash-based Message Authentication Code) qu'il positionne à la suite du message. Cela permet à l'entité qui reçoit le message d'authentifier et de vérifier l'intégrité du message, car il est nécessaire d'avoir la clé secrète pour créer ou vérifier le HMAC.
4. Chiffrement : le fragment compressé (ou non) est finalement authentifié et chiffré. Une fois le chiffrement effectué, le message doit être de taille inférieure à $2^{14} + 2048$ octets.

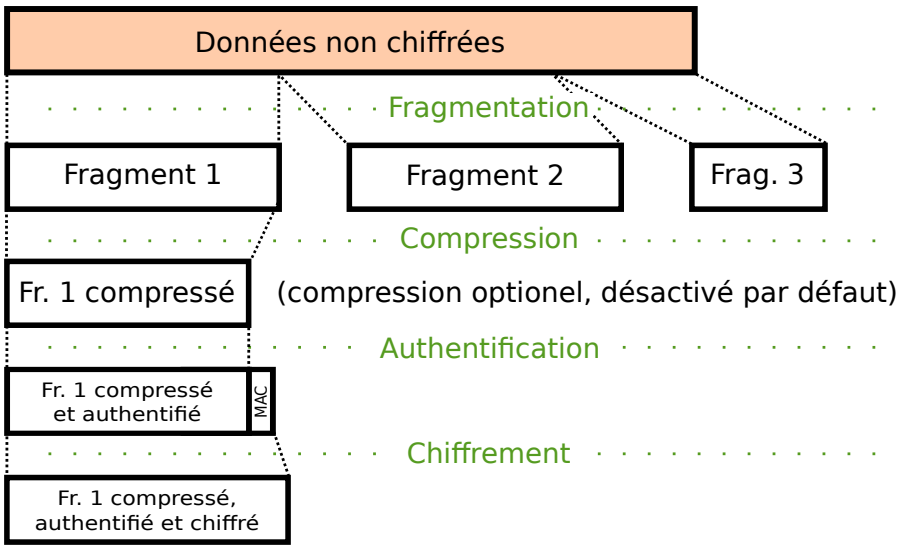


FIGURE 1.5 – TLS record layer

Comme nous l'avons vu dans la structure des records TLS, les entêtes du protocole sont accessibles en clair. Ainsi, en collectant le trafic entre un client et un serveur, il est possible de récupérer les entêtes des records TLS.

Dans les travaux présentés dans la suite de ce document, nous nous sommes intéressés spécifiquement au contenu des pages web. Ainsi, nous avons considéré uniquement les records qui contenaient des données : les records de type *APPLICATION DATA*. Ces derniers sont directement identifiables en repérant les entêtes possédant un champ *ContentType* égal à 0x17

(application_data). Pour ces records, nous pouvons lire le champ de l'entête longueur et ainsi avoir une estimation de la quantité de données envoyées. Comme expliqué précédemment, la compression étant désactivée par défaut, la différence de taille entre la taille lisible dans l'entête et la quantité réelle d'informations échangées est faible.

1.3 HTTP et son évolution

Dans cette section, nous présentons le protocole HTTP, qui est le protocole par défaut des navigateurs web afin de récupérer des données sur un serveur au moyen d'une URL [15]. Le standard HTTP n'est pour l'instant pas nécessairement sécurisé. Ainsi, dans le cas de l'usage de HTTP non sécurisé, les URL commencent par http :// puis sont suivies du nom de domaine ou adresse IP du serveur et du chemin de la page recherchée. HTTP est un protocole applicatif et a besoin du protocole TCP pour être transporté. Ce protocole HTTP est attribué par défaut au port 80 pour la version non sécurisée. Lorsque HTTP est sécurisé avec SSL/TLS, les URL débutent alors par https :// et le port par défaut est le port 443.

Conçu par Tim Berners-Le et al. entre 1989 et 1991 au CERN (Conseil européen pour la recherche nucléaire), le protocole HTTP a été inventé conjointement avec le format des pages web (HTML [16]), le premier navigateur web (WorldWideWeb) et un serveur web (httpd). Plusieurs versions, standardisées ou non, du protocole ont vu le jour pour donner aujourd'hui ce qu'on appelle communément le web. Ces versions sont décrites ci-dessous.

1.3.1 HTTP/0.9 : version initiale

Le numéro de version 0.9 correspond bien à la première version du protocole HTTP. Elle fut appelée ainsi a posteriori pour la différencier des versions ultérieures.

Cette version de HTTP est extrêmement rudimentaire. Une requête de type *GET* est envoyée par le client, qui s'est au préalable connecté au serveur via TCP. Suite à la réception de ce message, le serveur envoie directement le fichier HTML demandé.

Il n'y avait pas de code d'erreur et on ne pouvait envoyer que des documents de type HTML.

1.3.2 HTTP/1 : extension du protocole

Différentes améliorations de HTTP/0.9 ont été mises en place entre 1991 et 1995 de façon non organisée. Ainsi, afin d'unifier ces extensions et de rendre interopérables les différents clients et serveurs, la RFC 1945 fut publiée en 1996. Comme indiqué sur la RFC, cette dernière n'est pas un standard mais une note informelle. Cependant, elle définit des mécanismes bien connus aujourd'hui, comme par exemple :

— Les codes HTTP : information, succès, redirection, erreurs client ou erreurs serveur.

— Les entêtes HTTP : pour les requêtes et les réponses. Cela permet la transmission de métadonnées et rend ainsi le protocole plus flexible.

— Les méthodes : HEAD et POST.

On constate que, parallèlement aux avancées pour HTTP, la nécessité de sécurité se fait ressentir, car le réseau n'est plus limité au cadre académique. Dès 1995, on voit émerger l'usage d'une

15. Uniform Resource Locator ou adresse web

16. HyperText Markup Language : ce langage est utilisé pour représenter le contenu et la structure d'une page web

couche supplémentaire dans le transport du protocole HTTP. En effet, en plus des protocoles classiques TCP/IP, on peut trouver le protocole de chiffrement SSL.

1.3.3 HTTP/1.1 : standardisation

La standardisation de HTTP débute dès 1995. Elle a été publiée en 1997 dans la RFC 2068 [9] sous le numéro de version HTTP/1.1. Ce nouveau standard permet de résoudre les problèmes d'interopérabilité et apporte également des améliorations. On citera :

— Les connexions persistantes.
— La négociation automatique de l'encodage, du type de document et de la langue entre le client et le serveur.
— L'amélioration du contrôle sur le cache.
— La création de l'entête Host, qui permet l'hébergement de plusieurs domaines via des entités virtuelles sur une même adresse IP.
— La création du pipelining (expliqué dans le paragraphe 1.4.2).

Cette version a été utilisée majoritairement pendant une quinzaine d'années, avec toutefois deux révisions du standard avec la RFC 2016 [10] en 1999 et les RFC 7230-7235 [11] en 2014, jusqu'à l'arrivée et l'adoption de HTTP/2.

1.3.4 HTTP/2 : simplification et performances

Les pages web sont devenues avec le temps de plus en plus complexes, et sont aujourd'hui des applications. Avec HTTP/1.1 pour rendre le chargement rapide, il est nécessaire d'ouvrir plusieurs connexions TCP en parallèle et de synchroniser l'ordre de chargement des objets. A cause de ces contraintes, Google définit dès 2010 un nouveau protocole : SPDY [12]. L'objectif était de réduire la latence en augmentant la réactivité des échanges client-serveur et en réduisant la quantité de données redondantes transmises. Google amène également avec SPDY une nouvelle vision de la sécurité de HTTP et impose l'usage du chiffrement.

Avec les résultats et le succès de SPDY, ce protocole a servi de base pour la construction du protocole HTTP/2. Suite à la publication officielle du standard HTTP/2, le protocole SPDY a été retiré pour être remplacé définitivement par HTTP/2. Nous présentons ci-dessous quelques nouvelles fonctionnalités apportées par HTTP/2 :

— L'encodage du protocole se fait désormais en binaire.
— Les entêtes HTTP/2 sont compressées.
— HTTP/2 introduit le multiplexage (cf. paragraphe 1.4.2).
— La technique "Server push" permet au serveur HTTP/2 d'envoyer des données au client avant que ce dernier n'ait envoyé la requête correspondante.

La RFC 7540 [1] couvrant le standard HTTP/2 a été publiée en mai 2015. Son adoption fut rapide grâce à la réduction des coûts engendrés par son usage, qui diminue le trafic. Actuellement, sur l'ensemble des pages web du net, 45% des pages [17] sont servies avec le protocole HTTP/2. Cependant, la majorité du trafic est en version HTTP/2 si l'on considère le volume. En effet, dès avril 2016, 68% du trafic utilisait le procotole HTTP/2 d'après keycdn [18]. Or, à cette époque, moins de 11% des pages web supportaient HTTP/2. Cela s'explique par l'adoption de HTTP/2 par des services très populaires comme Youtube ou Netflix. Ces derniers consomment une très grande partie de la bande passante et sont aussi largement utilisés.

17. `https://w3techs.com/technologies/details/ce-http2`
18. `https://www.keycdn.com/blog/http2-statistics`

1.3.5 HTTP/3 : une révision en profondeur

Afin de toujours optimiser les temps de chargements des pages web, cette nouvelle version de HTTP change en profondeur le fonctionnement du protocole. En effet, HTTP/3 change de protocole de transport et passe de TCP à QUIC [13]. QUIC est un protocole de transport multiplexé et sécurisé, utilisant le protocole UDP pour le transport et TLS 1.3 pour le chiffrement. La structure interne à HTTP/3 réutilise en grande majorité celle de HTTP/2 (les frames).

Actuellement (juin 2020) HTTP/3 [14] est encore en discussion à l'IETF et a encore le statut de *draft*. Cependant le protocole possède déjà plusieurs implémentations dans les navigateurs modernes (Chrome, Firefox et Safari) mais n'est pas encore activé par défaut sur ces derniers. D'après le W3techs [19], on constate tout de même que début juillet 2020, plus de 6% des sites web supportent HTTP/3.

Dans la figure 1.6, nous synthétisons les différentes couches protocolaires possibles en fonction de la version de HTTP avec chiffrement.

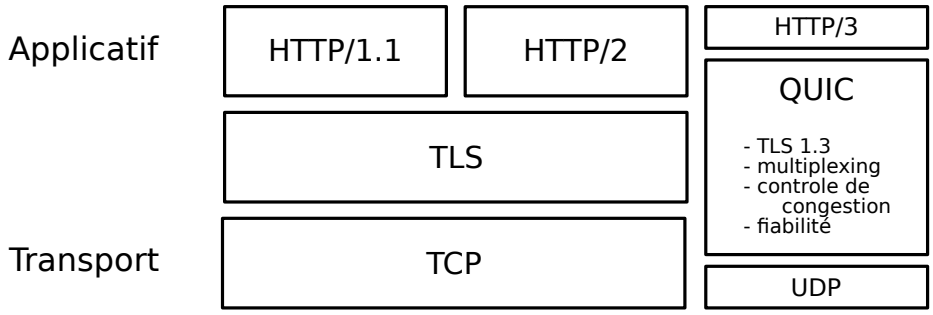


FIGURE 1.6 – Couches protocolaires en fonction des versions de HTTP (avec chiffrement)

1.4 HTTP/1.1 et HTTP/2 : détail et comparaison

Dans cette section, nous détaillons et comparons les deux versions HTTP/1.1 et HTTP/2 que nous avons plus particulièrement étudiées lors de nos travaux. Nous décrivons l'établissement d'une connexion, les modèles d'échange des données et le format des messages HTTP.

L'ensemble des informations de cette section sont issues des RFC 7230-7235 (HTTP/1.1) et 7540 (HTTP/2).

1.4.1 Etablissement de la connexion

Avant de récupérer des données avec le protocole HTTP sur un serveur distant, il est nécessaire d'établir une connexion avec ce dernier. Les différentes étapes à effectuer afin d'arriver aux premiers octets de données échangés sont détaillées dans ce paragraphe. L'ensemble des informations concernant l'établissement d'une connexion HTTP pour les versions 1.1 et 2 est résumé dans la figure 1.7.

19. `https://w3techs.com/technologies/details/ce-http3`

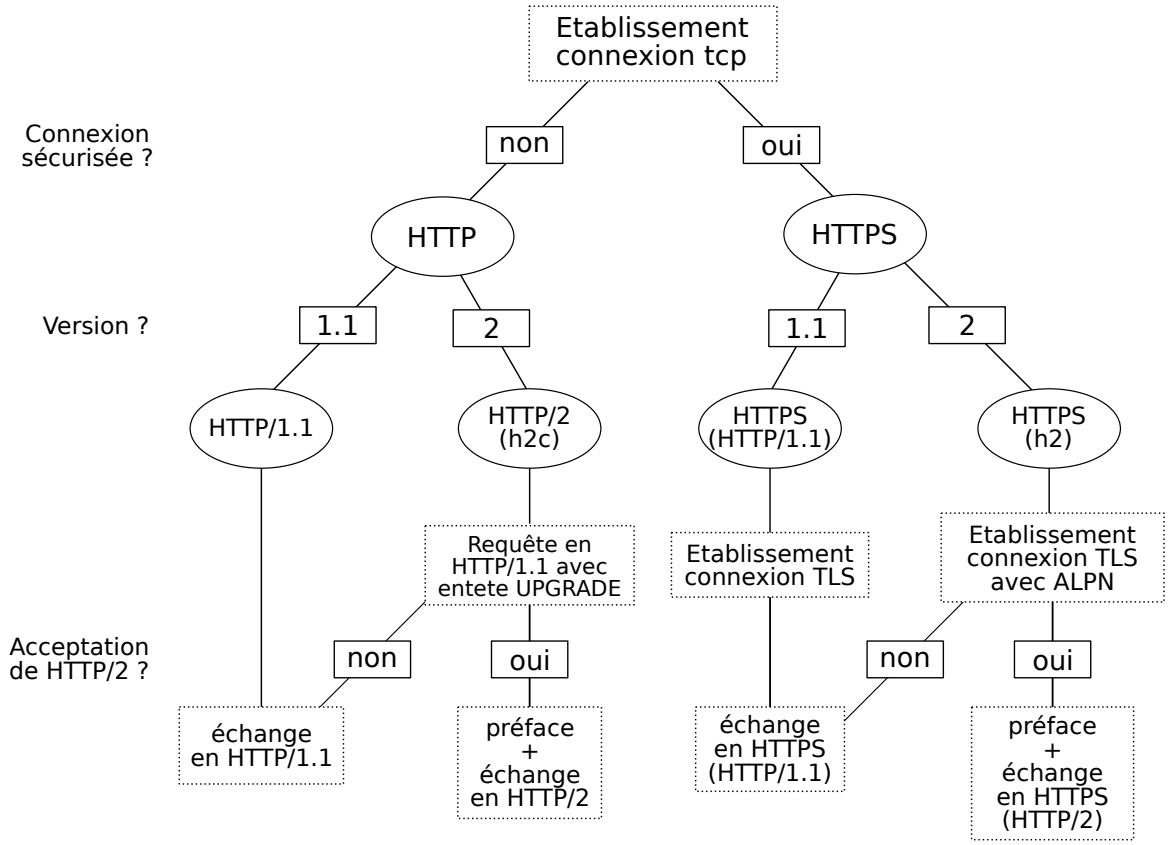


FIGURE 1.7 – Établissement d'une connexion HTTP/1.1 ou HTTP/2

Transmission Control Protocol (TCP)

HTTP est un protocole construit sur les couches TCP/IP, l'établissement de la connexion entre le client et le serveur passe donc, pour débuter, par l'établissement d'une connexion TCP classique. La mise en place de la connexion s'effectue en trois étapes : l'envoi d'un message SYN par le client, puis le serveur répond avec un SYN-ACK et finalement le client renvoie un paquet ACK.

Après l'établissement de la connexion TCP, l'échange peut être ou non sécurisé.

HTTP non sécurisé

Pour HTTP/1.1 c'est très simple, il suffit que le client envoie un message de requête dans le bon format comme défini plus tard dans le paragraphe 1.4.3.

En revanche pour le protocole HTTP/2, il est nécessaire pour le serveur et le client d'envoyer un message spécifique avant l'envoi des données. Ce message s'appelle *préface de connexion HTTP/2* et correspond à la chaîne de caractères suivante : "PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n".

Il n'est cependant pas nécessaire pour le client ou le serveur d'attendre le message de préface de l'autre entité pour commencer l'envoi de données (frame HEADERS et DATA, cf. paragraphe 1.4.3).

Lorsque le client ne sait pas si le serveur supporte le protocole HTTP/2, le client peut envoyer sa requête au format HTTP/1.1 avec une entête "Upgrade : h2c". Le sigle "h2c" fait référence à "HTTP/2 clear" (en clair) et s'oppose à "h2" qui rend implicite l'usage de HTTP/2 sécurisé. Le serveur peut alors répondre comme si l'entête n'était pas là s'il ne supporte pas HTTP/2, sinon il envoie une préface de connexion HTTP/2 avant de répondre à la requête avec le format de message HTTP/2.

HTTPS (HTTP sécurisé)

Suite à l'établissement de connexion TCP, le client peut effectuer une demande d'établissement de session SSL/TLS (client Hello). Pour le protocole HTTP/1.1, le client peut envoyer son message de requête dès la fin de la mise en place de la connexion sécurisée. Il s'en suivra un échange classique de requêtes/réponses dans le tunnel chiffré.

Pour le protocole HTTP/2, afin de débuter directement l'échange avec cette version, une extension appelée "Application-Layer Protocol Negotiation" (ALPN) est normalisée dans la RFC 7301 [15] en parallèle de HTTP/2. Cette extension permet au client de déclarer dans le "Client Hello" (début de mise en place d'une connexion TLS) son intention de communiquer en utilisant le protocole HTTP/2. Le serveur répond dans le message "Server Hello" le protocole qu'il a sélectionné parmi les propositions du client. Si HTTP/2 est sélectionné, à la fin de la mise en place de la connexion SSL/TLS, le client enverra la "préface de connexion HTTP/2" et démarrera son échange normalement.

1.4.2 Les modèles d'échanges

A l'origine avec la version 1 de HTTP, chaque connexion était utilisée pour un unique échange : une requête puis une réponse. Afin d'améliorer les performances, les versions ultérieures reposent sur différentes méthodes pour réduire et accélérer les échanges.

Dans la suite de ce paragraphe nous ferons successivement référence aux sous-images de la figure 1.8 qui rassemblent l'évolution des échanges HTTP avec les différentes versions et options.

HTTP/1.1

Avec HTTP/1.1 la principale amélioration est la mise en place de connexions persistantes. Cela signifie qu'après l'établissement de la connexion TCP (+ TLS si sécurisé), plusieurs échanges requête/réponse sont possibles sur une même connexion. Il n'y a plus besoin d'initier une nouvelle connexion à chaque fois.

Respectivement les sous-figures 1.8(a) et 1.8(b) représentent les échanges avec HTTP/1 et HTTP/1.1. On constate aisément le gain de temps en évitant l'ouverture d'une nouvelle connexion pour chaque requête/réponse.

Cependant afin de réduire encore plus les temps de chargement un nouveau mécanisme a été mis en place : le pipelining. Ce dernier mode d'échange permet au client d'envoyer plusieurs requêtes en même temps au serveur sans que ce dernier n'ait pour autant fini de répondre à la première requête. Ainsi on économise un aller/retour de paquet car le serveur peut directement traiter les requêtes à la suite. Ce modèle d'échange est schématisé dans la sous-figure 1.8(c).

Le pipelining est plus efficace que la simple connexion persistante en théorie mais en pratique son utilisation a été très limitée et même stoppée à cause de problèmes de compatibilité et

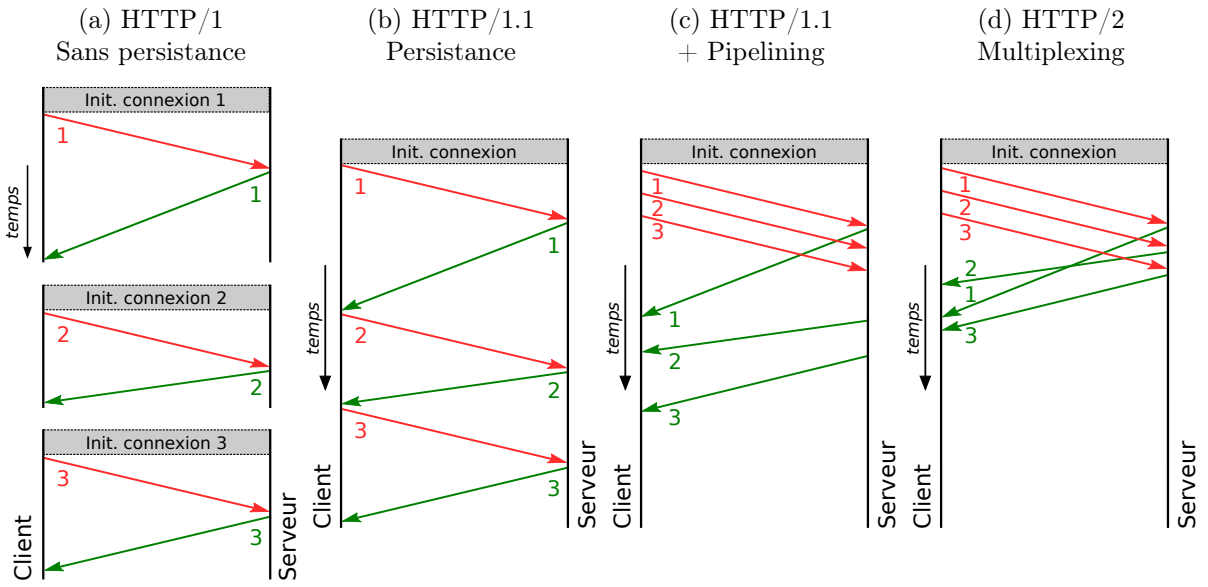


FIGURE 1.8 – Récapitulatif des différents modes d'échanges de HTTP

de certaines limitations comme le blocage en tête de ligne[20] (Head Of Line blocking). Cette limitation apparaît lorsque le traitement d'une requête côté serveur est bloqué. Dans ce cas, l'ensemble des requêtes en attente déjà envoyées par le client sont à l'arrêt et ne peuvent être annulées. Dans les faits, le pipelining a été intégré dans les principaux navigateurs mais n'a pas été activé par défaut. De plus dés le début 2017 le support de cette option a été simplement supprimé[21] [22].

Ainsi dans la suite de nos travaux, lorsque nous faisons référence à du trafic HTTP/1.1, nous ne considérons pas l'usage du pipelining.

Pour améliorer les performances, les navigateurs ont recours à l'ouverture de plusieurs connexions TCP en parallèle afin de permettre le chargement de plusieurs objets en même temps. Bien que dans les faits la solution soit fonctionnelle, cela complexifie les développements des applications web pour lesquelles la synchronisation des différentes connexions est fondamentale. C'est une des raisons du développement du protocole HTTP/2, qui apporte une réponse à ce problème.

Une des conséquences du modèle d'échange de HTTP/1.1 sans pipelining (cf. figure 1.8(b)) fait qu'il est possible de reconstruire la taille d'un objet HTTP chiffré (entête + message envoyé par le serveur). En effet, il suffit d'observer la quantité de données échangée entre deux messages en provenance du client (deux requêtes) pour obtenir cette taille. Le chiffrement ne modifiant pas significativement la taille des messages, cette dernière donne une information très proche de ce que l'on pourrait récupérer sans chiffrement.

HTTP/2

Avec HTTP/2, le modèle d'échange s'oriente encore un peu plus vers l'amélioration de la performance. A présent pour HTTP/2 il ne doit y avoir qu'une seule connexion ouverte entre

20. `https://community.akamai.com/customers/s/article/How-does-HTTP-2-solve-the-Head-of-Line-blocking-HOL-issue`
21. `https://www.chromium.org/developers/design-documents/network-stack/http-pipelining`
22. `https://bugzilla.mozilla.org/show_bug.cgi?id=1340655`

le client et le serveur. Cependant cette connexion est associée à plusieurs liaisons virtuelles qui peuvent communiquer en parallèle, on appelle cela le multiplexage.

Cette nouvelle notion de liaison virtuelle se nomme : stream. Chaque requête du client sera effectuée dans un stream différent et la réponse associée sera envoyée dans ce stream. Pour gérer ces streams, HTTP/2 repose sur un entête qui sera indiqué dans chaque message : "ID stream".

Le "ID stream" est unique pour chaque couple requête/réponse. Les ID impairs sont réservés pour les stream initialisés par le client et les ID pairs par le serveur [23]. L'ID est incrémenté pour chaque nouveau stream jusqu'à atteindre $2^{31} - 1$ pour une connexion. Si l'ID max est atteint, une nouvelle connexion doit être initiée.

Avec le multiplexage le client peut envoyer plusieurs requêtes en parallèle et le serveur peut également traiter plusieurs réponses en parallèle. Cette méthode s'oppose avec le principe FIFO mais permet ainsi d'éviter les problèmes de blocages en tête de ligne car si une réponse est bloqué le serveur peut envoyer les réponses associées à d'autres requêtes.

1.4.3 Le format des messages

Suite à l'établissement de la connexion qui a été présenté dans le paragraphe 1.4.1, le protocole HTTP consiste en un échange de messages entre un client qui fait une demande de contenu et un serveur qui distribue du contenu. On distingue alors deux types de messages : les requêtes en provenance du client et les réponses en provenance du serveur.

Afin de garder un maximum de compatibilité avec la version 1.1, HTTP/2 est structuré de façon similaire à HTTP/1.1 bien que le format soit un peu différent. Nous commençons donc par présenter le format du protocole HTTP/1.1.

HTTP/1.1

Cette version du protocole est au format texte ASCII comme les anciennes versions. Chaque message est structuré comme indiqué dans la figure 1.9. CRLF correspond à la séquence retour chariot (CR) et saut de ligne (LF). Les éléments précédés d'un astérisque sont facultatifs.

Message	=	Ligne-d'amorçage *(Entêtes du message CRLF) CRLF *(Corps du message)
Ligne-d'amorçage	=	Ligne-requête ou Ligne de statut

FIGURE 1.9 – Structure générique d'un message pour HTTP/1.1

La ligne de requête est, comme son nom l'indique, spécifique pour les requêtes et la ligne de statut est associée aux réponses. Les entêtes ont un format identique pour les deux types de messages. Toutes les entêtes suivent le format *<clé> : <valeur>* et chaque entête se situe sur une ligne indépendante. Certaines entêtes sont spécifiques aux requêtes, d'autres aux réponses

23. Le serveur peu ouvrir un stream lorsqu'il utilise les *push_promise*, que nous détaillerons par la suite

ou communes aux deux. On trouve par exemple des entêtes pour le contrôle du cache, de la connexion ou simplement pour dater la génération du message. Il y a également des entêtes pour la description du contenu selon qu'il provient du client ou du serveur. On citera par exemple les entêtes "Content-Type" ou "Content-Length" qui définissent respectivement le type du contenu envoyé ou sa longueur.

Les requêtes :

La première ligne d'une requête est composée de trois éléments séparés par des espaces :

— La méthode : elle correspond à une méthode de requête qui permet différents types d'action en fonction de la méthode invoquée. La liste des méthodes peut être étendue mais on retrouve souvent certaines méthodes standard comme : GET, HEAD, POST, DELETE, PUT, etc. Cependant seules les méthodes GET et HEAD sont nécessairement supportées par le serveur web.

— Une URL : elle désigne la ressources recherchée.

— La version HTTP : dans notre cas "HTTP/1.1.

Dans les entêtes des requêtes on trouve par exemple :

— Accept-Encoding : définit l'encodage/compression attendu par le client pour la ressource demandée. Comme pour le protocole TLS, il existe une attaque présentée en 2013 : BREACH [16], qui peut donner accès à des informations malgré le chiffrement lors de l'usage de la compression. Il est donc déconseillé d'utiliser de la compression lors de la transmission de données sensibles.

— Accept-Language : définit les langues acceptées par le client.

— User-Agent : indique le contexte du client qui a généré la requête. Un navigateur donnera par exemple son nom et sa version, ce qui permettra au serveur d'adapter sa réponse.

— Referer : indique au serveur l'URL depuis laquelle on a effectué la requête courante.

Les réponses :

Concernant les réponses, la première ligne est également composée de trois éléments séparés par des espaces. Elle débute par la version HTTP (HTTP/1.1). Ensuite nous avons le numéro de code du statut à trois chiffres puis une phrase descriptive du statut.

Les codes de statut sont séparés en cinq catégories et les numéros de code commencent respectivement par 1 pour les informations, 2 les succès, 3 les redirections, 4 les erreurs du client et 5 les erreurs du serveur. La liste des erreurs standards avec leur description précise est disponible à l'adresse suivante : `https://httpstatuses.com/`.

Enfin concernant les entêtes spécifiques aux réponses, on peut citer :

— Age : indique le temps écoulé depuis la génération de la donnée. Cela permet par exemple de contrôler la pertinence des éléments qui sortent du cache.

— Location : indique où se situe la ressource demandée. Cet entête est utilisé dans le cas d'une redirection.

— Server : donne des informations sur le serveur comme le fait le user-agent pour le client.

HTTP/2

HTTP/2 change par rapport aux versions antérieures et devient binaire. L'ensemble des entêtes du protocole ne sont plus au format texte. Cela rend le protocole difficilement compréhensible par les humains mais comme la génération et la réception des requêtes était déjà systématiquement automatisée, le format binaire facilite ces traitements.

Tous les messages HTTP/2 possèdent maintenant une structure comme celle décrite dans la figure 1.10.

— La taille du message : cette dernière donne la taille du message sans considérer les entêtes.

— Le type de frame : il existe différentes structures pré-définies de frame. La structure et le contenu du corps du message dépendent complètement du type de frame. Certaines frames sont détaillées juste en dessous.

— Les flags : ces flags booléen sont spécifiques à chaque type de frame.

— R : un bit réservé qui doit être maintenu à zéro lors de l'envoi et ignoré à la réception.

— L'identifiant du stream : il permet à HTTP/2 de multiplexer les connexions TCP, comme expliqué dans le paragraphe 1.4.2

— Le corps du message de la taille définie par le premier entête. Son format dépend du type de frame annoncé.

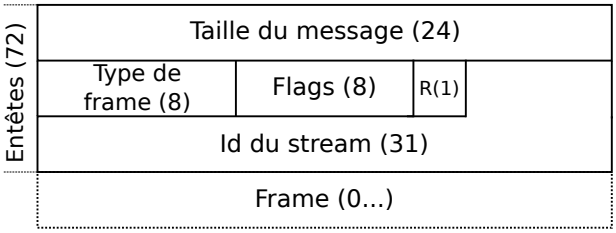


FIGURE 1.10 – Structure HTTP/2 (tailles en bits)

Les Frames :

Il existe dix types de frames qui ont chacune une structure binaire différente. Cela signifie que chaque type de frame possède son ensemble d'entêtes propres. Nous allons présenter les trois types de frames principaux (cf. section 6 de la RFC 7540 [1] pour avoir le détail complet de toutes les frames), qui remplissent des fonctionnalités attendues pour HTTP ou apportent des nouveautés avec HTTP/2. La structure de ces trois types de frames est schématisée dans la figure 1.11.

— HEADERS : c'est la structure qui contient les entêtes des messages (requêtes ou réponses). Ce type de frame permet l'initialisation d'un nouveau stream. Pour les requêtes, il rassemble l'ensemble des entêtes sous la forme *<clé> : <valeur>* comme pour HTTP/1.1 mais il est nécessaire d'ajouter trois nouveaux entêtes : " :method", " :scheme" et " :path" qui remplacent la première ligne des requêtes pour HTTP/1.1.

Pour les réponses il est obligatoire d'ajouter l'entête " :status" qui définira le code de statut pour la réponse.

Une fois que la liste des entêtes est définie, HTTP/2 compresse les entêtes. En effet HTTP/2 a mis en place un système de compression des entêtes : HPACK (RFC 7541 [17]) qui a été

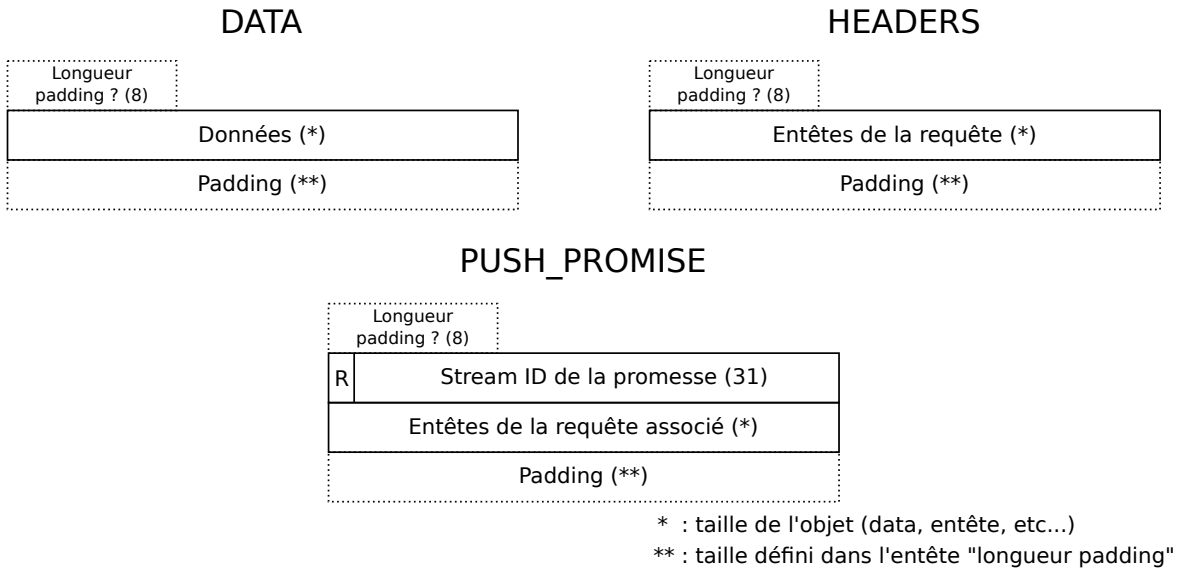


FIGURE 1.11 – Exemple de frames (tailles en bits)

conçu pour être résistant aux attaques sur du contenu compressé et chiffré, comme les attaques CRIME et BREACH.

— DATA : cette structure contient les données qui sont envoyées. La structure DATA est utilisée principalement pour les réponses mais également pour les requêtes POST par exemple. La structure DATA est toujours précédée d'une frame HEADERS ou PUSH_PROMISE.

— PUSH_PROMISE : ce type de frame est associé à la notion de push en provenance du serveur. C'est un nouvelle fonctionnalité de HTTP/2 qui permet au serveur d'envoyer du contenu au client avant que ce dernier n'en ait fait la requête. Ce système permet de réduire les temps de chargement d'une page lorsque cette dernière a plusieurs dépendance connues par le serveur.

  Par exemple lorsque le serveur reçoit une requête d'une page HTML, le serveur envoie le contenu sur le même stream que la requête et également des frames PUSH_PROMISE correspondant aux fichier .css et .js relatifs à cette page. Suite à l'émission de ces frames, le serveur renvoie sur des streams séparés chaque élément en débutant par une frame HEADER (réponse) suivie d'une frame DATA avec le fichier correspondant.

  Dans la structure, R est un bit réservé, le stream ID de la promesse permet au serveur d'annoncer sur quel stream il va envoyer les données et les entêtes correspondent à la requête qui génère les données envoyées par le serveur.

Pour l'ensemble des types de frames, il y a possibilité d'effectuer du padding directement au niveau de HTTP. Il suffit d'activer le flag padding pour ajouter une entête de 8 bits qui définira la longueur du nombre d'octets de padding à ajouter à la suite des données.

1.5 Synthèse

HTTPS ou “HyperText Transfer Protocol Secure” est la combinaison des protocoles HTTP et SSL/TLS. HTTP est sécurisé par SSL/TLS qui est lui-même transporté par le protocole TCP. Dans ce chapitre nous avons présenté en détail les protocole SSL/TLS et HTTP.

Le protocole de chiffrement SSL/TLS protège les messages en les chiffrant et utilise une structure de données nommée *record* (cf. figure 1.5). Cette structure fonctionne avec des entêtes disponible en clair donnant la version du protocole, le type de record et la taille (cf. figure 1.4). Nous avons présenté, ici, le fonctionnement de la version 1.2 de TLS.

Concernant, le protocole HTTP, il existe sous différentes versions. Nous nous focalisons sur les deux versions 1.1 et 2 de HTTP car ce sont celles qui sont les plus utilisées actuellement. HTTP/1.1 est l'ancienne version majeure de HTTP qui est petit à petit remplacé par HTTP/2 depuis 2015. HTTP est un protocole central via son usage dans les navigateurs web, ce qui en fait un sujet d'étude privilégié.

Le modèle d'échange des messages ainsi que la structure de ces derniers sont significativement différents entre HTTP/1.1 et HTTP/2. Ce chapitre a expliqué les différents modèles d'échanges ainsi que l'établissement de la connexion pour ces différentes versions du protocole. Chaque connexion HTTP/1.1 est une alternance d'échange de requêtes/réponses alors que HTTP/2 introduit le *multiplexing* et permet la gestion en parallèle de plusieurs requêtes/réponses sur une même connexion. Lorsque ces protocoles sont protégés par TLS et que l'on parle de HTTPS, il est important de noter que les traces générées par ces deux protocoles ne seront pas semblables.

Chapitre 2

Les méthodes d'analyse de trafic chiffré

Sommaire

2.1 Introduction 30
2.2 Détection de protocoles 30
2.2.1 Trafic TLS 31
2.2.2 Trafic HTTPS 32
2.3 Détection de services 32
2.3.1 Utilisation d'informations en clair 33
2.3.2 Website fingerprinting 34
2.3.3 Mobile service classification 36
2.4 Détection d'actions utilisateur sur un service HTTPS 36
2.4.1 Solutions actives 36
2.4.2 Solutions passives 37
2.5 Synthèse 38

2.1 Introduction

L'analyse du trafic à des fin de classification est un domaine très vaste largement couvert par la communauté scientifique, comme en attestent les différents états de l'art existant dans le domaine : [18, 19, 20, 21, 22, 23, 24, 25]. Cependant, seuls quelques uns de ces travaux se sont intéressés au cas spécifique du trafic chiffré avec différentes granularités de classification : P. Velan *et al.* [23] (2015), W. Shbair *et al.* [24] (2017) et F. Pacheco *et al.* [25] (2019).

La figure 2.1 schématise ces différents niveaux avec l'exemple du service Google :

— Niveau protocole : le but est d'associer un flux à un protocole (exemple : Skype, HTTP, POP3, Bittorent, SSH, etc...) ou à un type d'activité (exemple : browsing, chat, mail, VoIP, game, etc). Dans le cas de HTTPS, on a deux niveaux de protocole : TLS pour le chiffrement et HTTP pour les données.

— Niveau service : après avoir reconnu le protocole, on peut s'interroger sur le nom du service qui est utilisé. Par exemple, avec HTTP, on cherche à définir la page web contactée (nom de domaine).

— Niveau actions utilisateur : on cherche maintenant à identifier un comportement de l'utilisateur au sein d'un service (exemple : recherche d'un mot, clic sur un bouton, etc). C'est à ce niveau que nos travaux s'inscrivent.

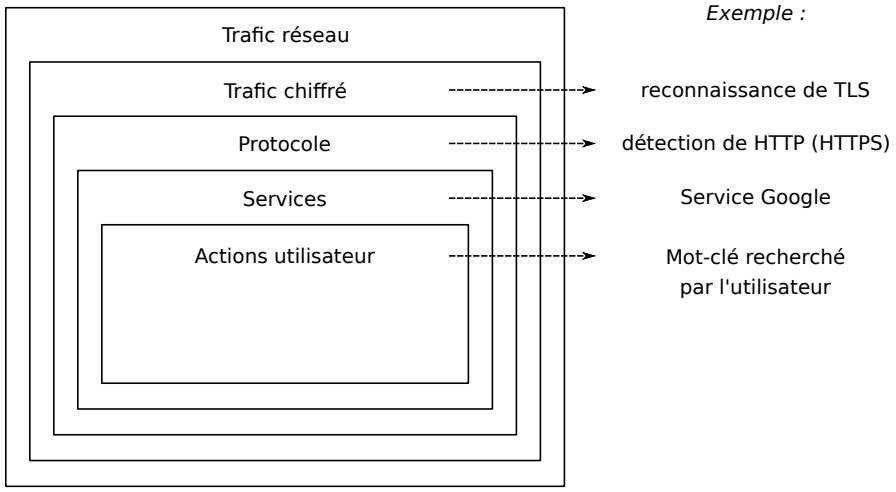


FIGURE 2.1 – Découpage des différents niveaux de classification

Dans la discussion sur la classification de trafic chiffré de P. Velan [23] (2015), l'identification du service chiffré, en particulier pour le protocole TLS, est considéré comme l'enjeu majeur. En suivant cette considération, nous présentons dans la suite, les travaux qui cherchent à classifier le protocole TLS et/ou HTTPS, ainsi que ceux qui identifient les services sous-jacents. En complément de cela, nous étudions aussi le dernier niveau de granularité qui se situe au niveau du comportement d'un utilisateur dans son usage d'un service et qui constitue l'objet de nos recherches.

2.2 Détection de protocoles

Cette section présente les travaux sur la détection du protocole TLS, et plus particulièrement celle de HTTPS (HTTP sécurisé avec TLS). La majorité des travaux de recherche ne se

focalisent pas uniquement sur ces deux protocoles, mais visent aussi, par exemple, à identifier du trafic SSH [26, 27, 28, 29], ou du trafic Skype [30, 31, 32].

Comme décrit dans [24], ces travaux se scindent généralement en trois catégories principales : l'analyse des ports TCP/UDP, l'analyse de la structure des messages du protocole, et les solutions d'apprentissage automatique sur un ensemble de caractéristiques du trafic réseau. Les deux paragraphes suivants se focalisent sur la détection de TLS puis de HTTPS. Il est important de noter pour la suite que les travaux sont difficilement comparables en termes de résultats quantitatifs, car ils utilisent des jeux de données différents.

2.2.1 Trafic TLS

La méthode triviale pour identifier un protocole applicatif est d'observer le port TCP ou UDP utilisé. Il est alors possible d'associer le protocole à ce port en utilisant la table des ports definie par une autorité officielle représentée par l'IANA [24] (Internet Assigned Numbers Authority). Il est important de noter que cette liste n'est qu'indicative, et donc certains services peuvent être configurés pour utiliser des ports arbitraires, et donc non officiels.

On peut constater qu'il n'y a pas de port par défaut pour le protocole TLS puisque il est lié au service applicatif (HTTPS : 443, SMTP(sécurisé) : 465, LDAPS : 636, IMAPS : 993, etc.). Cependant, ces règles ne sont pas toujours respectées, certains serveurs HTTPS sont ainsi liés au port 8080 et, à l'inverse, des services comme Bittorent utilisent le port 443 pour éviter le filtrage par port sans nécessairement utiliser le protocole TLS. Dans [33], les auteurs montrent qu'entre 1,1% et 5% des applications utilisent un port attribué à des services au dessus de TLS sans utiliser ce protocole. A l'inverse, pour un maximum de 4,2% des cas (pour leur jeu de données) des services protégés par TLS utilisent d'autres ports que ceux normalement réservés. De la même façon, [34] montre les limites de l'usage des ports en étudiant différents jeux de données qui utilisent cette information pour construire la vérité-terrain.

Ainsi, le port TCP associé à la connexion que nous essayons d'identifier donne une indication quant à la nature du protocole, mais il nécessaire d'utiliser d'autres méthodes d'identification complémentaires pour pouvoir statuer de façon certaine.

Afin d'identifier un protocole, il est également possible de s'appuyer sur la structure des messages. En effet, comme nous l'avons vu dans la section 1.2.2, le protocole TLS, a une structure de messages bien définie. Cette méthode utilise donc des techniques dites « de DPI »(inspection profonde des paquets) pour accéder à la charge utile des paquets et reconnaître leur structure. On retrouve la présentation de cette technique dans plusieurs travaux [33, 35, 36, 37] qui cherchent à reconnaître la structure de *record* en regardant les premiers octets (entrant et/ou sortant) de chaque connexion. Les différences entre ces travaux résident dans le nombre d'octets considérés et le nombre de paquets étudiés. [33] cherche à valider la reconnaissance du premier message *Server-Hello* et donne un taux de justesse pour la détection de TLS de 80%. [35] tente de reconnaître le format d'un record TLS (cf. figure 1.4) en observant les cinq premiers octets de chaque paquet et obtient 95% de détection pour TLS. Dans [37], les auteurs atteignent 99% en utilisant les quatre premiers records d'une connexion.

Il existe également des solutions OpenSource comme PACE [38], nDPI [39], NBAR [40] ou Libprotoindent [41] qui permettent de faire de la classification. Les deux études [19, 42] comparent ces approches et s'accordent sur les bonnes performances de ces solutions. On peut noter par

24. `http://www.iana.org/`

exemple que la solution nDPI a détecté 100% des connexions TLS dans [19]. Globalement, l'analyse de la structure des messages est une solution très performante pour reconnaître le trafic TLS.

Enfin, concernant les méthodes d'apprentissage automatique pour la détection spécifique du protocole TLS, nous citons les travaux de C. McCarthy [43], qui étudient exclusivement la détection entre du trafic TLS et du trafic non TLS. Cette étude montre la possibilité d'utiliser l'apprentissage supervisé pour apprendre à reconnaître du trafic TLS. Les auteurs comparent plusieurs algorithmes, dont AdaBoost, C4.5 et RIPPER. Les évaluations montrent que AdaBoost donne les meilleurs résultats avec un taux de justesse de 95,7%.

Malgré les bons résultats observés, l'usage de l'apprentissage automatique n'est pas une solution privilégiée pour la détection de TLS. En effet, cette solution est plus complexe à mettre en place car elle nécessite une collecte de traces d'apprentissage et une analyse complète de chaque connexion. De plus, elle produit des résultats a priori légèrement inférieurs aux méthodes d'analyse de la structure des messages.

2.2.2 Trafic HTTPS

Contrairement au protocole TLS, HTTPS a un port normalisé dédié (443). Mais là encore, bien d'autres applications peuvent communiquer via ce port, de façon légitime ou non, comme nous l'avons énoncé précédemment. Pourtant, on constate dans la littérature que cette méthode est très souvent utilisée pour labelliser le trafic HTTPS dans des jeux de données : [44, 45, 46, 47], bien que ce ne soit pas une technique extrêmement fiable. La particularité de HTTPS (HTTP + TLS) est qu'il est transporté et sécurisé par le protocole TLS. N'ayant pas accès aux entêtes du protocole HTTP, il n'est donc pas possible d'utiliser la détection de structure du protocole afin de reconnaître du trafic HTTPS.

De fait, les méthodes d'identification pour HTTPS sont essentiellement orientées vers la reconnaissance de caractéristiques issues du trafic sous-jacent suite à la détection préalable de l'utilisation du protocole TLS via sa structure de message. Les auteurs de [33] utilisent du clustering (semi-supervisé) sur la taille des paquets de données et reconnaissent HTTPS à plus de 99,9%. Q. Sun et al. [37] utilisent de l'apprentissage supervisé via l'algorithme Naive Bayes couplé à huit caractéristiques statistiques sur la taille des paquets, le temps inter-paquets, la durée du flux et le nombre de paquets. Leur solution permet de reconnaître 93,13% des connexions HTTPS. Enfin, dans [48] les auteurs analysent uniquement les 64 premiers octets d'une connexion TCP en tant que vecteur de caractéristiques pour catégoriser le trafic. Leur solution utilise le principe d'entropie maximum [49] pour définir une signature du protocole HTTPS et atteint un taux de classification de 99%.

2.3 Détection de services

Nous nous intéressons à présent aux travaux qui effectuent de la classification de trafic avec une granularité plus fine. Plus précisément, nous étudions l'ensemble des méthodes existantes pour détecter les services accédés avec le protocole HTTPS. Dans le cas de trafic issu d'un navigateur, un service correspond à une page web, tandis que pour du trafic mobile, les services sont les applications ayant généré le trafic.

Nous ferons mention, dans la suite, de travaux qui considèrent des scénarios dits "en monde fermé" ou "en monde ouvert". Cela fait référence à deux types de scénarios pour mesurer l'efficacité d'une solution de classification qui utilise de l'apprentissage :

— Soit l'évaluation est réalisée uniquement avec des traces correspondant à des services rencontrés pendant la phase d'entraînement. Les scénarios sont alors dits en monde fermé.

— Soit l'évaluation est réalisée avec des traces correspondant à des services non connus. On parle alors de scénarios dits en monde ouvert.

Dans le cadre d'un scénario en monde ouvert, l'évaluation est plus complexe car l'algorithme de classification doit être capable de reconnaître si le modèle de la trace a été appris ou non avant de classer précisément cette dernière si c'est le cas. Pour un scénario en monde fermé cette étape n'existe pas.

Cette section se décompose en trois parties. La première couvre les solutions qui utilisent de l'information disponible en clair pour repérer l'usage d'un service. Dans le second paragraphe, nous présentons les travaux sur le “website fingerprinting” (prise d'empreinte de site web) dont l'objectif est de détecter l'accès à des pages web utilisant HTTPS, mais également protégées par un VPN ou une solution d'anonymisation. Finalement, nous terminons avec l'étude des travaux concernant la détection des applications mobiles.

2.3.1 Utilisation d'informations en clair

Une première solution triviale pour détecter des services est d'utiliser les adresses IP à la manière des ports TCP pour les protocoles. Cependant, comme pour les ports, la fiabilité n'est pas garantie. En effet, en fonction de la localisation d'un utilisateur ou de la répartition des charges réseaux, un service peut être accédé depuis différentes adresses IP. Tenir une liste à jour pour chaque service est de plus en plus complexe. A cela s'ajoute la mutualisation des hébergements : plusieurs services sont fréquemment hébergés avec une unique adresse IP.

Le DNS est également une solution intéressante car, avant d'accéder à un service, le navigateur fera auparavant une requête DNS (contenant le nom de domaine) pour récupérer l'adresse IP du service ciblé. En revanche, là aussi, ce procédé n'est pas fiable car un utilisateur peut faire une requête directement avec une adresse IP. La résolution DNS peut aussi être chiffrée avec DoH [50] (DNS sur HTTPS) ou DoT [51] (DNS sur TLS), être effectuée localement ou être en cache, rendant l'accès à cette information impossible. On pourrait également utiliser des outils comme le DNS inversé mais les données ne sont pas nécessairement à jour.

Il existe d'autres informations disponibles en clair afin d'identifier les services. Lors du *handshake* TLS, le serveur envoie son certificat au client afin de l'authentifier. Le certificat intègre le nom de domaine du service contacté, ce qui permet ainsi d'identifier simplement le service. [35] implémente cette solution et utilise également le *session ID* de TLS qui permet de suivre l'ouverture de nouvelle connexion avec ce service, même dans le cas où le certificat n'est pas échangé. Cette méthode n'est pour autant pas infaillible car avec la mutualisation des services, un certificat peut être associé à plusieurs services différents, rendant l'identification précise très difficile. De plus, on peut noter que la prochaine version de TLS (TLS 1.3) chiffre l'envoi du certificat, rendant l'accès à cette information impossible.

Dans les entêtes TLS, on trouve également le champ *Server Name Indication* (SNI) qui indique le nom de l'hôte que l'on souhaite contacter. [52] utilise cette information combinée avec le certificat pour filtrer l'usage de services. Cependant, [53] montre les limites du filtrage via

le SNI, en citant deux problèmes : la rétrocompatibilité (ce champ n'est pas obligatoire et ne compromet pas le *handshake*, même si le nom du SNI n'est pas valide), et le partage de certificats inter-services (on peut par exemple accéder au service Youtube de Google en utilisant le SNI du service Google Maps).

Ainsi, on constate que les solutions qui se construisent sur les informations disponibles en clair peuvent potentiellement identifier le service. Pour autant, chacune des ces méthodes est aisément contournable rendant leur utilisation peu fiable.

2.3.2 Website fingerprinting

Le website fingerprinting, ou la détection de pages web par empreinte, est une catégorie de méthodes pour détecter le chargement de pages web malgré la présence de solutions de chiffrement. Historiquement, le chiffrement en question était souvent un VPN (réseau privé virtuel) ou un tunnel SSH. Actuellement, les méthodes de chiffrement ont évolué et le website fingerprinting aussi, et nous observons à présent dans l'état de l'art des solutions qui supervisent du trafic anonymisé.

Dès 1996, des travaux commencent à mettre en cause certaines garanties du chiffrement pour protéger complètement la vie privée des utilisateurs face à l'analyse de trafic. En effet, D. Wagner [54] met en avant l'existence d'une vulnérabilité de SSL par rapport à cela : *"[SSL 3.0] offer only minimal confidentiality protection [...]. The only change to SSL's protection against passive attacks worth recommending is support for padding to stop traffic analysis."*. Il s'ensuit, quelques années plus tard, plusieurs travaux. Cheng *et al.* [55] (1998), puis Sun *et al.* [56] (2002) ou encore Hintz *et al.* [57] (2002) qui montrent la faisabilité des attaques par analyse de trafic en étudiant uniquement le nombre et la taille des données transportées. Si Hintz *et al.* analysent du trafic utilisant un ancien proxy web sécurisé (SafeWeb), les deux autres approches reposent sur du trafic non chiffré mais n'exploitent pas le contenu des paquets comme s'ils étaient chiffrés.

Par la suite, les travaux liés au website fingerprinting s'intéressent à la reconnaissance des services visités malgré l'utilisation d'un VPN dans le cas d'un scénario en monde fermé ([58, 59, 60]). Liberatore *et al.* [59] (2006) enregistrent une séquence de tailles de paquets (ainsi que leurs directions) et l'associent avec le nom du site dont provient la trace d'entraînement. Les auteurs introduisent ensuite deux solutions permettant d'évaluer cette méthode. Celles-ci utilisent respectivement la distance de Jaccard ou la classification naïve bayésienne qui calcule une distance entre la trace en cours d'analyse et l'ensemble des traces d'entraînement. Lu *et al.* [58] (2010) étudient l'usage du tunnel OpenSSH comme solution VPN et construisent des signatures en observant la séquence des tailles des requêtes HTTP ainsi que la séquence des tailles des réponses (en excluant les tailles correspondant au MTU [25] des paquets). Les auteurs annoncent une amélioration de la détection de 11% avec cette nouvelle solution par rapport à celle de Liberatore. Les travaux de Liberatore sont également repris par Herrmann en 2009 qui les compare avec sa nouvelle solution [60] reposant sur une classification naïve bayésienne multi-nominale (cf. [61] p. 258) couplée avec un vecteur de caractéristiques contenant la fréquence d'apparition des tailles de paquets. En réalisant les tests avec des traces protégées avec un tunnel OpenSSH (4 traces pour 775 services différents), les solution de Liberatore ont des taux de justesse entre 82% et 86%, alors que Herrmann atteint 94%. Dans ces même travaux, les auteurs évoquent que la considération d'un scénario en monde ouvert serait plus réaliste. De plus, leur solution est testée

25. Maximum Transmission Unit (tcp)

face aux solutions d'anonymisation, mais des taux de détection faibles contre les solution TOR (2,96%) et JAP (19,97%) sont alors obtenus (ces solutions sont définies ci-dessous). Une suite d'études s'est ainsi focalisée sur le trafic des solutions d'anonymisation dans le cadre de scénarios en monde ouvert.

Les solutions d'anonymisation (PA : proxy d'anonymisation) les plus courantes sont le réseau TOR [26] [62], I2P [27] [63] ou JonDonym [28] (anciennement JAP). Montieri & al. ont développé une solution [64] permettant de distinguer l'usage de ces trois solutions d'anonymisation avec de très bons résultats (99,99% de justesse). Les auteurs utilisent un ensemble de caractéristiques issues de l'outil Tranalyzer2 [65], dont par exemple des statistiques sur les temps inter-paquets, la longueur de ces derniers ou la durée des connexions. Leur approche teste quatre algorithmes de machine learning : Naïve Bayes, Bayesian Network, C4.5 et Random Forest, qui donnent tous de très bons résultats. Elle utilise le jeu de données Anon17 mis à disposition par K. Shahbar & al. [66], qui a collecté des traces réseau provenant des trois PA énoncés avec une granularité plus fine encore car il permet également de retracer le type de trafic ainsi que les services visités.

Tout d'abord, en 2011, Pachenko *et al.* [67] mettent au point une attaque contre le navigateur TOR qui n'exploite pas la taille des paquets puisque TOR utilise du padding pour unifier toutes ces tailles. Ainsi, en utilisant la taille du document principal, le ratio paquets entrant/sortant, le nombre de paquets et les *burst* de paquets, les auteurs passent de 3% à 55% de taux de justesse pour le même jeu de données que [60]. En plus de cela, ils considèrent un nouveau jeu de données afin d'avoir un scénario en monde ouvert et obtiennent un taux de vrais positifs de 73% (0,05% de faux positifs). X. Cai *et al.* [68] reprennent le principe des séquences de tailles [59] en considérant également la direction (positif : paquet entrant, négatif : paquet sortant) mais en appliquant simplement la distance de Damerau-Levenshtein [69, 70] utilisée habituellement pour calculer une distance entre deux chaînes de caractères. L'évaluation a été effectuée dans le cas d'un scénario en monde fermé sur 100 pages web différentes et les auteurs ont obtenu un taux de justesse de plus de 80%. En 2014, Wang *et al.* [71] montrent une nouvelle attaque utilisant l'algorithme des k plus proches voisins (kNN) couplé à un ensemble de caractéristiques issues des précédents travaux dans le domaine. Cette classification donne 85% de justesse dans le cas d'un scénario en monde ouvert pour la détection de 100 pages.

Dans [72], les auteurs mettent en avant l'importance d'avoir un jeu de données représentatif du trafic Internet non limitatif aux pages d'accueil comme le font la plupart des autres travaux. Cette volonté de confronter les études du domaine à des situations plus réalistes fait suite à la publication de M. Juarez [73] qui met en lumière plusieurs lacunes constatées dans la littérature sur le website fingerprinting. Ainsi, les auteurs de [72] proposent une méthode (CUMUL) reposant sur l'algorithme SVM couplé à des caractéristiques qui suivent l'évolution du nombre de paquets entrant ou sortant dans le temps, ainsi que celle de leur taille cumulative.

Enfin, J. Hayes *et al.* [74], présentent une nouvelle solution nommée k-fingerprinting, qui utilise un ensemble de 150 caractéristiques et l'algorithme des forêts d'arbres décisionnels (RF). Ils comparent leur approche avec [72] et [71]. Face à du trafic TOR classique, ces trois solutions ont des résultats similaires avec un taux de justesse d'environ 91%. Cependant, k-fingerprinting donne des résultats plus solides quand différentes techniques de défense [75, 76, 67, 77, 78] sont mises en place.

26. `https://www.torproject.org/`
27. `https://geti2p.net/en/`
28. `https://anonymous-proxy-servers.net/index.html`

2.3.3 Mobile service classification

Un autre pan important relatif à la détection de services concerne le trafic mobile pour la détection d'applications. Les premiers travaux dans le domaine remontent à 2013 avec la solution NetworkProfiler [79] qui s'articule autour de l'inspection de paquets (DPI), ainsi que l'enregistrement des adresses IP contactées par les applications, et ne supporte donc pas le trafic chiffré. [80] utilise des métriques semblables et propose d'utiliser des proxys de déchiffrement pour gérer le cas du trafic chiffré et pouvoir continuer les méthodes autour du DPI. [81] met en lumière que, dans le trafic mobile, seulement 30% du trafic est généré par l'utilisateur directement ; le reste provient de trafic en arrière plan souvent régulier et possédant des motifs reconnaissables pour les différentes applications installées. Ainsi, en observant uniquement les motifs dans le trafic mobile chiffré, on peut repérer si certaines applications sont installées sur un téléphone. [82] considère l'algorithme Random Forest avec un ensemble de caractéristiques du trafic utilisant la taille des paquets et le temps inter-paquets. Leur classification permet de reconnaître 13 applications IOS avec un taux de détection supérieur à 85% via l'interception du réseau sans fil chiffré.

V.F. Tayloar [83, 84], via AppScanner, regroupe à la fois un outil pour générer et classifier les signatures des applications. La solution est testée sur un large ensemble de 110 applications Android avec une classification binaire par application. Sur cet ensemble, la solution donne un taux de détection de 99% en utilisant l'algorithme Random Forest. En 2018, G. Aceto [85] met en place un framework qui va utiliser en parallèle différentes méthodes issues de travaux précédemment présentés : [60, 59, 83, 86]. Les auteurs montrent que les résultats obtenus en combinant ces différentes méthodes sont meilleurs que chacun pris indépendamment et ce sur plus de 90 applications mélangeant IOS et Android. Les même auteurs publient en 2019 la solution MIMETIC [87], qui s'articule autour du deep learning. En plus d'utiliser le deep learning pour réaliser leur classification, les auteurs insistent sur le réalisme de leur jeu de données, obtenu via la capture de trafic réel mélangeant IOS et Android.

2.4 Détection d'actions utilisateur sur un service HTTPS

Dans cette dernière section, nous présentons les travaux et outils disponibles pour détecter un comportement d'un utilisateur au sein d'un service. Nous séparons ces travaux en deux catégories : les solutions actives qui altèrent le trafic d'une quelconque façon et les solutions passives qui reposent uniquement sur l'écoute trafic.

2.4.1 Solutions actives

Une des solutions très fréquemment utilisée est le proxy de déchiffrement. Il est placé entre un client et un serveur avec lequel le premier souhaite communiquer. Le client passe par le proxy pour communiquer avec le serveur cible. C'est le proxy qui initie la connexion avec le serveur et sert de passerelle d'échange entre les deux entités, comme le ferait une attaque par homme du milieu. On peut citer comme exemple des solutions telles que Apache [29], Nginx [30] ou Squid [31]. Le flux étant non chiffré au niveau du proxy, toutes les solutions existantes utilisant de l'inspection de paquets peuvent alors s'appliquer. Il existe néanmoins plusieurs désavantages à une telle solution, notamment l'abandon de la protection de la vie privée, car le proxy accède à l'ensemble des données personnelles en clair ; une couverture d'utilisation limitée car la machine utilisateur doit

29. `https://httpd.apache.org/`
30. `https://nginx.org/`
31. `http://www.squid-cache.org/`

être configurée pour utiliser un tel service (par exemple en réseau d'entreprise). D. Zakir montre dans [88] que ces proxys sont également souvent mal configurés et introduisent potentiellement des failles de sécurité qui affaiblissent la protection des données. Il faut ainsi suivre de bonne pratique pour éviter ces problèmes. En France, l'agence nationale de la sécurité des systèmes d'information (ANSSI) a par exemple publié une note technique [89] donnant les recommandations de sécurité pour éviter ces problèmes de configuration dés 2014.

Afin de ne pas avoir à faire confiance à un tiers pour filtrer le trafic, une autre solution active a été proposée par J. Sherry avec la solution Blindbox [90]. Elle vise à renforcer la sécurité des outils d'interception de trafic pour appliquer des règles de filtrage sans que l'outil effectuant la détection n'ait connaissance du contenu du trafic. Pour réaliser cela, les auteurs s'appuient sur du chiffrement recherchable [91, 92]. Comme illustré dans la figure 2.2, on peut voir qu'en plus du canal de communication classique SSL/TLS, un second canal de données est mis en place. Ce dernier découpe les données par mot (token) puis les chiffre avec du chiffrement recherchable. Cette méthode de chiffrement permet de chiffrer des données tout en permettant de faire une recherche de mots-clés sans avoir à déchiffrer le document dans son ensemble. C'est sur ce principe que la sonde peut détecter l'usage de certains mots-clés dans le flux. Cette solution souffre cependant d'une lenteur conséquente à l'initialisation de chaque connexion (dépendant du nombre de règles). D'autres travaux comme [93, 94] se sont intéressés à cette approche et ont repris le principe tout en améliorant significativement les performances.

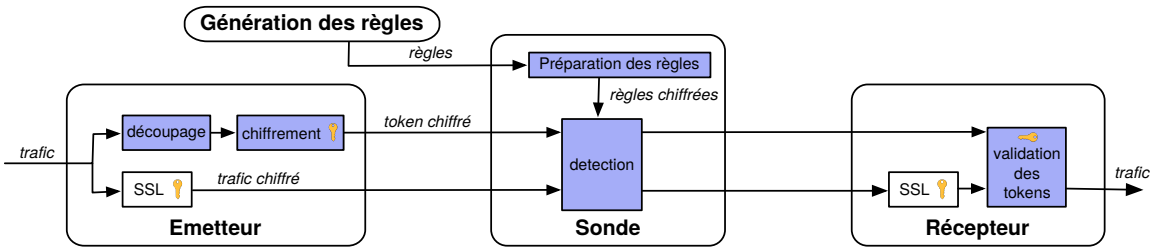


FIGURE 2.2 – Schématisation de la solution Blindbox
(d'après J. Sherry [90])

2.4.2 Solutions passives

Les solutions qui visent à détecter des comportements spécifiques au sein d'un service sont principalement des applications prévues pour des situations très spécifiques. Par exemple, dans le domaine de la vidéo, les auteurs de [95] ont développé une méthode spécifique au flux vidéo chiffré de Netflix, leur permettant de reconnaître la lecture de différents films via la supervision du trafic de diffusion pendant quelques minutes. Dans de précédents travaux [96], les auteurs ont montré que la combinaison d'un taux d'échantillonnage variable (VBR) et l'utilisation de l'outil DASH (Dynamic Adaptive Streaming over HTTP) de streaming de Microsoft Silverlight permet la création d'une séquence de tailles de segments vidéo unique par contenu. Ainsi, cette séquence constitue une signature pour la vidéo associée. De façon similaire avec l'encodage audio (VoIP chiffré utilisant VBR [32]), C. Wright et al. ont pu reconnaître dans [97] la langue des participants

32. `http://www.linphone.org`

en utilisant uniquement la taille des paquets chiffrés. Pour 14 des 21 langues testées, les auteurs ont obtenu un taux de justesse supérieur à 90%. Dans [98], ces même auteurs ont approfondi ces travaux en détectant des portions de phrases dans du trafic VoIP lorsque ce dernier utilise le protocole SRTP [99].

Une autre catégorie de ce type de travaux correspond à la détection d'activités pendant l'utilisation de certaines applications mobiles. Les travaux de M. Conti [100] permettent la détection de 5 à 11 activités pour 7 applications et s'articulent autour des séries temporelles. J. Liu [101] utilise les caractéristiques issues des travaux décrits dans [102] et les applique pour la détection de 8 activités (3 applications). Enfin, B. Saltaformaggio [103] utilise des caractéristiques telles que le temps inter-paquets, le décompte du nombre de paquets ou leurs tailles et la fréquence d'apparition de certaines fenêtres de tailles de paquets. Sa solution est évalué sur 1 à 3 activités pour 22 applications. L'ensemble de ces solutions utilisent de l'apprentissage supervisé : SVM ou RF.

Enfin, la dernière catégorie de travaux s'articule autour de la reconnaissance de motifs dans le trafic. S. Coull *et al.* [104] ont appris à reconnaître la langue utilisée dans un flux de messages textuels provenant de l'application imessage. Cela est effectué grâce à un apprentissage préalable des statistiques des distributions de tailles de messages en fonction du langage et à une détection de ces statistiques dans du flux réel. L'étude du rythme de la frappe au clavier a également été utilisée dans [105] pour reconnaître l'envoi de commandes SSH. En effet, les auteurs ont constaté que chaque appui de touche générait un paquet TCP et ont fait une analyse statistique des temps moyens entre l'appui des différentes touches d'un clavier. Suite à cela, comme la simple observation du temps inter-paquets est équivalente à l'observation du temps entre l'appui des touches d'un clavier, les auteurs sont capables de reconnaître des commandes ou même des mots de passe. IOACTIVE LABS [106] produit en 2012 une démonstration permettant de reconnaître la localisation en cours de consultation d'un utilisateur sur le service Google Maps. Pour permettre cela, les auteurs ont préalablement appris la signature du trafic générée par le chargement des zones géographiques à superviser. Lors de la réalisation de ces travaux, il y a quelques années, Google Maps envoyait un fond de carte découpé en quadrillage générant plusieurs dizaines d'images ainsi qu'un deuxième quadrillage d'images à superposer avec le tracé des routes, les noms de lieux ou des commerces. C'est à partir de l'ensemble des tailles des paquets issues du chargement des différentes images de la carte que la signature des lieux a été apprise.

2.5 Synthèse

Par rapport à l'état de l'art, nous plaçons nos travaux au niveau de la détection des activités utilisateur en particulier pour le protocole HTTPS. Pour effectuer cela, notre solution doit reposer sur la reconnaissance de l'usage du protocole HTTPS et du service sous-jacent. Dans le présent état de l'art, nous avons pu montrer qu'il existe de nombreuses solutions fiables pour assurer la détection de ces deux niveaux de classification. Le tableau 2.1 recense l'ensemble des travaux précédemment cités utilisant de l'analyse de trafic. Comme on peut le voir la majorité des travaux s'intéresse à ces deux premiers niveau de classification.

La reconnaissance du port 443 est une méthode très fréquemment utilisée pour reconnaître l'usage du protocole HTTPS. Cependant, nous avons pu montrer que différentes méthodes plus fiables existaient pour effectuer ces détections. Ces méthodes reposent principalement sur la détection du protocole TLS via la structure des messages, suivie d'une deuxième classification pour

reconnaître HTTPS utilisant des caractéristiques de la connexion (statistiques sur les paquets ou informations extraites des entêtes TLS).

Concernant la détection du service transporté par HTTPS, il est possible d'utiliser les données présentes en clair mais comme précédemment expliqué, ces dernières ne sont pas forcement fiables. Ainsi, la méthode principale constatée consiste à extraire des caractéristiques du trafic et à utiliser une solution d'apprentissage supervisé pour apprendre un modèle contenant la signature des services que l'on souhaite détecter.

Pour finir, la détection au niveau de comportements utilisateur est globalement moins étudiée que les autres niveaux et est principalement dédiée à des applications très ciblées. Comme nous avons pu le voir, les travaux cités ci-dessus fonctionnent grâce à une particularité des services considérés. Globalement, la question de l'analyse de trafic pour réaliser des détections à ce niveau de raffinement n'a pas, à notre connaissance, été considérée pour être utilisée massivement sur plusieurs services.

Plusieurs travaux utilisent la taille chiffré des objets HTTP comme caractéristiques pour construire des signatures mais aucun travaux ne filtre ces tailles pour cibler des objets spécifiques afin d'effectuer de la classification de pages web. Nous avons appliqué cette technique dans le cadre de HTTP/1.1. Pour HTTP/2, il est important de noter qu'une grande partie des travaux est antérieure à l'usage de cette version du protocole ; en particulier pour les solutions relatives au website fingerprinting. En effet, le navigateur TOR est sélectionné très fréquemment pour les évaluations mais il utilise uniquement la version 1.1 de HTTP (couplé avec sa propre sur-couche de chiffrement).

Table 2.1 – Références triées par niveau de détection

Niveau de détection	Protocole	Service	Activité
Références	[64], [33], [37], [48], [44], [45], [46], [47], [43], [35], [36], [38], [39], [40], [41], [19], [42], [26], [27], [28], [29], [30], [31], [32]	[79], [80], [82], [83], [84], [85], [60], [59], [86], [87], [74], [72], [71], [68], [67], [107], [58], [55], [56], [57], [54], [52], [53], [35]	[101], [103], [102], [104], [105], [106]

Deuxième partie

Analyse de trafic chiffré HTTP/1.1

Chapitre 3

Modélisation et méthode pour l'analyse du trafic chiffré HTTP/1.1

Sommaire

3.1 Introduction 44
3.2 Modélisation du trafic chiffré HTTP/1.1 44
3.2.1 Modélisation initiale 44
3.2.2 Caractérisation d'une trace par des images 45
3.2.3 Modèle raffiné grâce à la détection de la taille des images chiffrées . 47
3.3 Méthode 48
3.3.1 Vue d'ensemble de notre solution 49
3.3.2 Extraction des caractéristiques 50
3.3.3 Génération des signatures 52
3.3.4 Algorithme de classification 57
3.4 Synthèse 59

3.1 Introduction

Comme expliqué dans l'introduction générale, notre objectif est de détecter des comportements utilisateurs illicites lors de l'utilisation d'un service HTTPS. Plus précisément, nous considérons un comportement utilisateur comme la recherche d'un mot-clé au sein d'un service sachant qu'un administrateur a défini une liste de mots-clés non légitimes au préalable. L'usage du protocole TLS par les services que nous considérons rend caduque l'inspection de paquet pour obtenir cette information. Pour pallier cette limitation, nous allons analyser les quantités d'informations échangées et leur direction pour reconnaître les motifs des comportements non légitimes. Par rapport à l'état de l'art nous proposons une méthode qui identifie un type d'objet HTTP échangé (les images) pour construire une signature performante dans le cas de l'utilisation de la version 1.1 du protocole HTTP avec chiffrement.

Le chapitre est découpé en deux sections. La section 3.2 décrit ainsi les différentes étapes qui nous ont amené à une modélisation fine prenant en compte les spécificités du trafic chiffré HTTP/1.1, associé à la recherche d'images par mot-clé. Malgré le chiffrement, nous sommes capables de récupérer la taille des objets HTTP chiffrés chargés au sein d'une page (cf. paragraphe 1.4.2) et en particulier les tailles des images. Nous constatons que les images sont un bon moyen de caractériser spécifiquement une page car elles sont souvent présentes en grand nombre avec des tailles variées, et cela nous permet d'en déduire, pour chaque page correspondant à un mot-clé, une signature.

Dans la section 3.3, nous détaillons notre méthode d'analyse qui se décompose principalement en deux parties : la génération des signatures et la phase de supervision. Les signatures sont générées sous la forme d'une fonction de densité via l'usage de l'algorithme de l'estimation par noyau (KDE pour Kernel Density Estimation). Pour la phase de supervision, nous calculons un score pour chacune des signatures précédemment générées afin de définir si la trace en cours d'évaluation correspond ou non à un mot-clé légitime.

3.2 Modélisation du trafic chiffré HTTP/1.1

Dans cette section, nous modélisons le trafic réseau associé au chargement d'une page sur un service donné lors de la recherche d'un mot-clé. Cette portion de trafic réseau est nommée "trace" ou "trace réseau" dans la suite de ce travail. Nous débutons par un modèle simple qui sera raffiné au fur et à mesure grâce aux différentes hypothèses considérées.

3.2.1 Modélisation initiale

Une trace réseau peut être vue comme une simple suite de paquets réseau et définie par l'équation suivante :

$$\forall a \in A, \exists m \in M : a = \langle m, \{p_1, \ldots, p_i, \ldots, p_n\} \rangle \tag{3.1}$$

A représente les traces réseau collectées et M l'ensemble des mots-clés possibles. Chaque élément a de A est une trace réseau qui correspond à un unique mot-clé m de M et est composée d'une séquence de paquets p_i.

Pour chaque paquet, la quantité d'information transportée utile est liée à sa charge utile (payload). Une solution simple est de récupérer la taille de la charge utile pour chaque paquet à partir des entêtes TCP (tcp.length - taille des entêtes). Cependant, dans le cas du trafic HTTPS,

TCP ne transporte pas directement le protocole HTTP car entre les deux se trouve le protocole TLS. Ce dernier possède également des entêtes en clair qui sont associées à une structure dite de "record". Cette structure est chiffrée mais les entêtes de TLS nous donnent accès à la taille en octets des records.

L'intérêt d'utiliser cette valeur vient de sa proximité avec la véritable information échangée. En effet, la quantité d'information à laquelle on peut accéder en clair est celle qui est la plus proche du chiffrement. Dans le paragraphe 1.4.3, nous avons expliqué la structure des records TLS et de leur entête. Ainsi, en utilisant les tailles de records TLS, il est possible de préciser l'équation (3.1). En notant r_i la taille du i-ème record TLS, la nouvelle équation s'écrit donc :

$$\forall a \in A, \exists m \in M : a = \langle m, \{r_1, \ldots, r_i, \ldots, r_n\}\rangle \tag{3.2}$$

Finalement, le problème posé dans l'introduction de ce chapitre revient à trouver la fonction f qui prend en paramètre une séquence de tailles de records TLS associé à un mot-clé m et reconnaît ce mot-clé si ce dernier est non légitime sinon il classe la trace comme légitime avec le label leg.

En notant M_{nl} l'ensemble des mots-clés non légitimes, on peut donc écrire le problème sous la forme suivante :

$$\begin{aligned} \forall a \in A, \exists m \in M : a =& \langle m, \{r_1, \ldots, r_i, \ldots, r_n\}\rangle \\ f(\{r_1, \ldots, r_i, \ldots, r_n\}) =& \begin{cases} m \in M_{nl} \\ leg \end{cases} \end{aligned} \tag{3.3}$$

3.2.2 Caractérisation d'une trace par des images

Les moteurs de recherche, sites de e-commerces ou encore réseaux sociaux, sont parmi les services les plus utilisés sur Internet. Ces services ont comme point commun de souvent proposer une barre de recherche qui permet de générer des pages précises liées à la recherche de l'utilisateur. Ces pages sont composées de contenus étroitement liés aux mots-clés qui ont été recherchés (images, texte, etc.). En effet, plus le résultat est proche de la requête, plus l'utilisateur est susceptible d'être satisfait. Ainsi, nous proposons de regarder si le contenu, et particulièrement les images dans les pages web, peut servir à produire une signature ou une empreinte représentative du mot-clé qui l'a générée.

Le nombre des images ou miniatures renvoyées par les pages n'est pas toujours identique. Cependant, sur des services tels que Amazon, DuckDuckGo Images, Instagram ou Google Images on constate qu'il y a en moyenne toujours plusieurs dizaines d'images. Les images ayant des résolutions différentes et sachant qu'elles sont également compressées (par exemple format jpeg), la taille des images varie fortement comme nous le verrons ci-après.

Nous devons d'abord évaluer si l'ensemble défini par les tailles des images contenues dans une page web peut servir à caractériser de manière unique cette page. Pour cela, il nous faut étudier la probabilité que deux pages possèdent le même ensemble de tailles d'images, c'est à dire la probabilité de collision des signatures générées.

Probabilité de collision de signatures

Le nombre d'images et la variabilité des tailles de ces images au sein d'un service sont les facteurs qui influent sur la probabilité de collision entre deux signatures. On note n le nombre

moyen d'images dans une page et T l'ensemble des tailles d'images possibles. Alors, le nombre de combinaisons possibles est $|T|^n$ (avec $|T|$ le nombre d'éléments dans T). En considérant un mot-clé et une distribution uniforme des tailles d'images, un deuxième mot-clé a une probabilité égale à $\frac{1}{|T|^n}$ d'avoir le même ensemble de tailles d'image et de causer une collision et donc une erreur d'identification.

Maintenant si on considère le problème pour un nombre k de pages différentes, cette situation se révèle être équivalente au paradoxe des anniversaires [108].

Paradoxe des anniversaires

Le paradoxe des anniversaires permet de calculer la probabilité que parmi k individus, deux d'entre eux aient la même date d'anniversaire. Il peut se généraliser comme suit :

Soit E un ensemble fini avec $|E|$ éléments, la probabilité $p(k)$ que, parmi k éléments de E, tirés de façon équiprobable, deux d'entre eux au moins soient identiques vaut :

$$\begin{aligned} P(k) &= 1 - \frac{|E|\ !}{(|E|-k)!} \times \frac{1}{|E|^k} \\ \text{or } P(k) &\approx 1 - e^{-\frac{k(k-1)}{2\times|E|}} \end{aligned} \tag{3.4}$$

Ainsi, en l'appliquant à notre cas d'images, on a $|E| = |T|^n$ et donc :

$$\begin{aligned} P(k) &= 1 - \frac{|T|^n\ !}{(|T|^n)^k \times (|T|^n-k)!} \\ \text{or } P(k) &\approx 1 - e^{-\frac{k(k-1)}{2\times|T|^n}} \end{aligned} \tag{3.5}$$

Cette probabilité repose sur l'hypothèse que les tailles ont une distribution uniforme, ce qui n'est a priori pas garanti.

Distribution des tailles des images

Dans la suite de ce paragraphe, nous expliquons l'expérience que nous avons mise en œuvre pour évaluer la distribution des tailles d'images (T) pour les quatre services suivants : Google Images, Duckduckgo Images, Instagram et Amazon. Nous avons collecté plusieurs milliers de traces liées à des mots-clés différents sous le format HAR (HTTP Archive). Ce format nous permet de récupérer en clair les informations sur les objets HTTP chargés. En particulier, il nous est possible d'extraire la taille et le type des objets rencontrés. Cela nous a permis d'évaluer le nombre moyen d'images chargées et la distribution des tailles d'images non chiffrées (en octet). La distribution cumulative des tailles d'images pour les quatre services est représentée sur le graphique de la figure 3.1. Sur cette figure, pour 60% des tailles observées pour chaque service, la distribution est quasi uniforme (entre les deux lignes pointillées). De plus, la portion de courbe uniforme est celle avec la pente la plus importante. En considérant uniquement les tailles d'images dans l'intervalle uniforme, on rend notre problème plus difficile car on perd de l'information mais cela nous permet d'appliquer l'équation (3.5).

En considérant Google Images, qui est le pire cas des quatre services de par sa plus faible variabilité des tailles d'images, nous observons en moyenne 50,58 images avec une déviation standard de 12,72. On observe que sur la section linéaire de la courbe, il y a environ $|T| = 6000$ tailles différentes possibles. En appliquant l'équation (3.5), la probabilité d'avoir au moins une collision

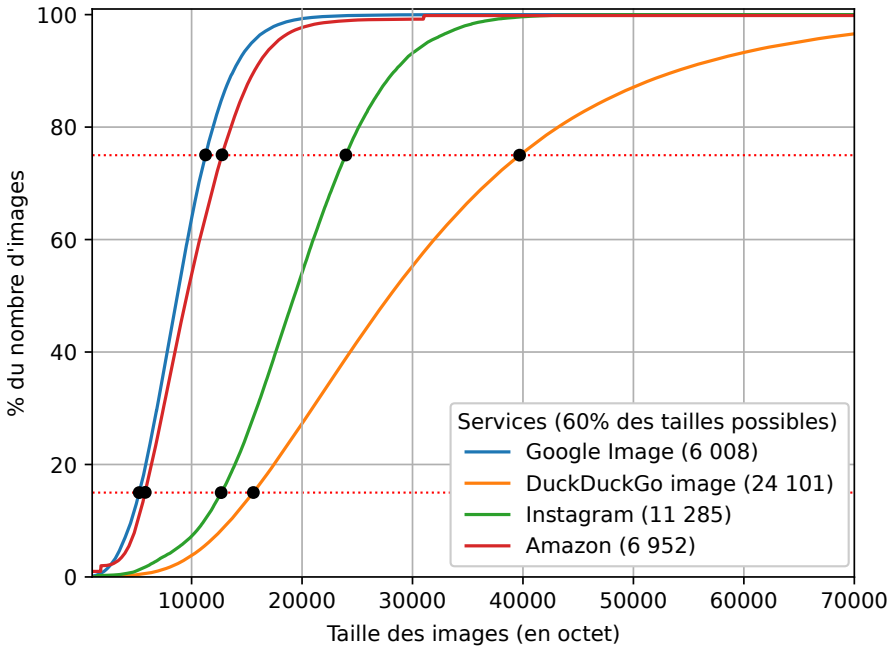


FIGURE 3.1 – Courbe cumulative du nombre d'images en fonction de la taille

avec 100 000 mots-clés est de $6,18 \times 10^{-180}$. Au vu de la probabilité très faible, cela montre bien que, d'un point de vue théorique, les tailles des images peuvent construire des signatures fiables. La pratique nous montrera les même résultats. Si on prend le dictionnaire anglais, qui contient $k = 230\ 000$ mots [33] et seulement 10 images sur une page ($n = 10$), la probabilité est de l'ordre de 10^{-28}, ce qui est encore très faible, et confirme l'applicabilité de la méthode.

Ainsi, la taille des images d'une page web, doit permettre d'identifier cette dernière de manière unique. Dans les paragraphes suivants, nous allons voir s'il est possible d'exploiter ce résultat lorsque les données sont chiffrées.

3.2.3 Modèle raffiné grâce à la détection de la taille des images chiffrées

Comme montré précédemment, l'utilisation de la taille des images est une caractéristique suffisante pour construire une signature. Cette dernière correspond à une page qui est elle-même associée à un mot-clé. Cependant, le chiffrement ne nous donne a priori pas accès à cette information pour deux raisons. Premièrement, la taille des objets HTTP en clair n'est pas disponible directement. Deuxièmement, il nous faut pouvoir identifier le type de l'objet pour ne garder que les tailles des images (et non celles des autres objets).

Taille des objets HTTP chiffrés

Dans le chapitre 1.4.2, nous avons montré qu'il était possible de reconstruire la taille des objets HTTP chiffrés dans le cas de l'utilisation du trafic HTTPS couplé avec le protocole HTTP/1.1 en observant les quantité de données émises par le serveur entre chaque paquet provenant du client. De plus, le chiffrement ne modifie pas la taille du contenu des paquets (aux octets de

33. `https://en.wikipedia.org/wiki/List_of_dictionaries_by_number_of_words`, 28/06/19

padding près dans le cas du chiffrement par bloc). Avec ces simples observations, nous sommes donc capables d'obtenir, pour toute trace réseau, la liste des tailles des n' objets HTTP chiffrés présents au sein de cette page.

Ainsi, on peut préciser le modèle de l'équation (3.2) comme suit :

$$\forall a \in A, \exists m \in M : a = \langle m, \{t_{O_1}, \ldots, t_{O_i}, \ldots, t_{O_{n'}}\}\rangle \tag{3.6}$$

Avec t_{O_i} la taille du i-ème objet HTTP chiffré contenu dans la trace a.

Filtrage des objets HTTP

Il nous faut maintenant filtrer les objets HTTPS pour ne garder que ceux liés à des images. En effet, cela permet d'éliminer les autres objets, souvent communs entre différentes pages associées à d'autres mots-clés (css, icônes, javascripts, etc.). Les éléments communs entre des pages liées à des mots-clés différents sont une source de bruit car ils n'aident pas à distinguer deux pages différentes. Ainsi, en les éliminant nous réduisons le bruit présent dans la signature. La technique que nous utilisons repose sur les tailles des requêtes associées aux objets HTTP chiffrés (envoyés dans les réponses). Elle permet de filtrer certains objets, en particulier les images à partir de ces tailles. Les détails du filtrage sont présentés dans le paragraphe 3.3.2.

Modèle raffiné

Finalement, pour chaque page web, nous pouvons récupérer la liste des tailles associées aux images chiffrées. Cette liste de tailles est utilisée en tant que signature pour un mot-clé car la probabilité que deux mots-clés aient la même signature est minime, comme nous l'avons vu avec le paradoxe des anniversaires.

Nous pouvons encore raffiner l'équation (3.6) en filtrant les objets HTTP pour ne garder que les n tailles d'images, nous modélisons nos futures traces réseau comme suit :

$$\forall a \in A, \exists m \in M : a = \langle m, \{t_1, \ldots, t_i, \ldots, t_n\}\rangle \tag{3.7}$$

Avec t_i la taille du chiffré de la i-ème image et $n < n'$.

Suite à cette dernière itération de la modélisation, notre problème initial est finalement de définir f' dans l'équation suivante :

$$\begin{aligned} \forall a \in A, \exists m \in M : a =& \langle m, \{t_1, \ldots, t_i, \ldots, t_n\}\rangle \\ f'(\{t_1, \ldots, t_i, \ldots, t_n\}) =& \begin{cases} m \text{ si } m \in M_{nl} \\ \text{sinon } leg \end{cases} \end{aligned} \tag{3.8}$$

3.3 Méthode

Pour rappel, notre solution prend en entrée une trace réseau correspondant au chargement d'une page suite à la recherche d'un mot-clé sur un service donné. En sortie, elle est capable d'inférer si le mot-clé fait partie de la liste des mots-clés légitimes ou non. Pour passer d'une trace réseau à la détection de mots-clés, nous utilisons une méthode statistique. Plus spécifiquement, nous utilisons une solution d'estimation de la densité qui considère des traces dites d'apprentissage afin de générer des signatures pour chaque mot-clé non légitime.

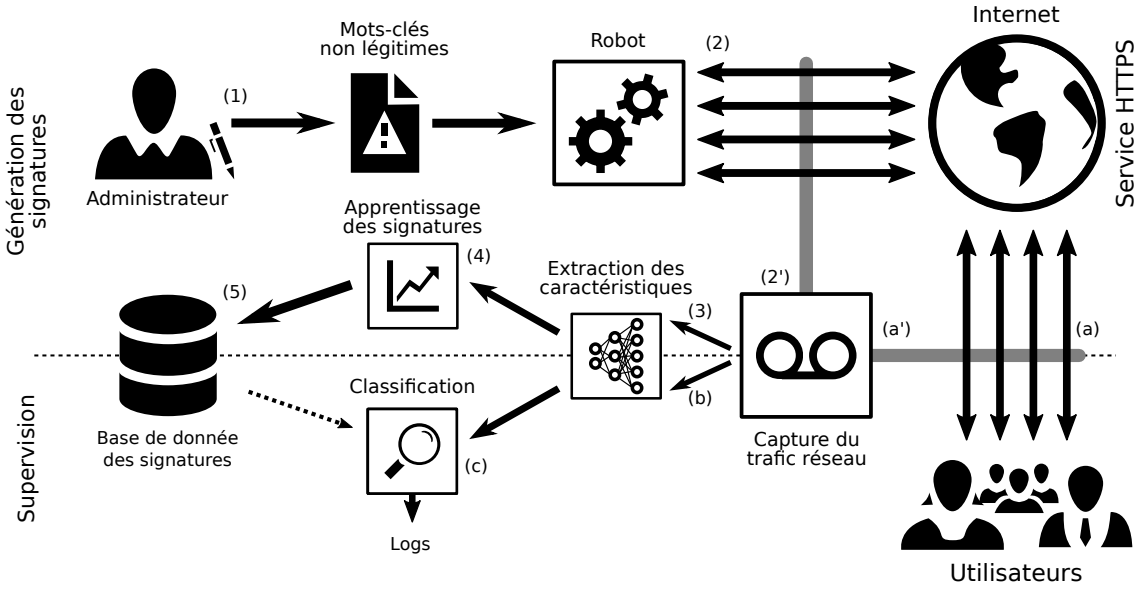


FIGURE 3.2 – Architecture globale de la solution (HTTP/1.1)

3.3.1 Vue d'ensemble de notre solution

La première partie de la solution, située dans la partie haute de la figure 3.2, correspond à la génération des signatures. Cette génération est découpée en plusieurs phases.

Pour débuter, (1) un administrateur définit une liste de mots-clés que la solution devra détecter. Cette liste de mots-clés, que l'on note M_{nl} (mots-clés non légitimes), contient les termes que l'administrateur du réseau considère comme illégaux ou contraires aux règles d'usage de l'infrastructure. Nous précisons ici qu'un mot-clé est considéré comme étant une expression composée d'un ou plusieurs mots du langage naturel.

Notre solution génère ensuite une signature à partir de traces réseau collectées pour chacun des mots-clés définis auparavant. Pour cela, un robot (2) va automatiser des requêtes correspondant aux mots-clés de M_{nl} sur le ou les services que l'on souhaite superviser. Lors des différents échanges entre le robot et les services cibles, (2') nous capturons l'ensemble des paquets réseaux. Ainsi, cela permet de former un ensemble de traces réseau pour chaque service et mot-clé concerné.

Lorsque les données ont été collectées pour chaque mot-clé non légitime, (3) la phase d'extraction des caractéristiques (les tailles des images chiffrées) peut débuter. Elle est présentée dans le paragraphe 3.3.2.

Suite à l'extraction, la phase (4) de génération des signatures a lieu. Elle permet de fusionner les informations acquises via différentes caractéristiques extraites précédemment sous la forme d'une fonction, que nous appelons signature. Cette étape est détaillée dans le paragraphe 3.3.3.

A la fin de cette phase, nous obtenons (5) une base de données des signatures correspondant aux mots-clés de M_{nl}.

La deuxième partie de la solution couvre le mécanisme de supervision. Lorsqu'un utilisateur fait une requête sur un service supervisé (a), le trafic réseau est automatiquement capturé (a') sous la forme d'une séquence de paquets réseau. A partir des traces réseau capturées, notre solution applique plusieurs traitements.

Nous procédons en premier lieu (b) à l'extraction des caractéristiques pour la classification. Cette étape est similaire à l'étape (3) de la phase d'apprentissage.

Notre solution (c) évalue ensuite ces caractéristiques au regard des différentes signatures (étape (5)) et détermine si la trace réseau est en accord avec la politique mise en place par l'administrateur ou non. La phase de classification est expliquée dans le paragraphe 3.3.4. Elle se décompose en un calcul de score suivit d'un filtrage, afin de distinguer si un résultat est représentatif d'une requête légitime ou non.

Selon le résultat obtenu, l'administrateur est libre de mettre en place le système de réponse de son choix. Il peut par exemple définir un système d'alerte, de log ou de blocage de trafic d'un utilisateur.

3.3.2 Extraction des caractéristiques

Comme expliqué précédemment, nous utilisons la taille des images comme attribut caractéristique pour chaque trace. Cette étape d'extraction permet de convertir une séquence de paquets réseau en une séquence de tailles d'images chiffrées pour une trace donnée. La figure 3.3 résume schématiquement les différentes étapes de cette phase d'extraction.

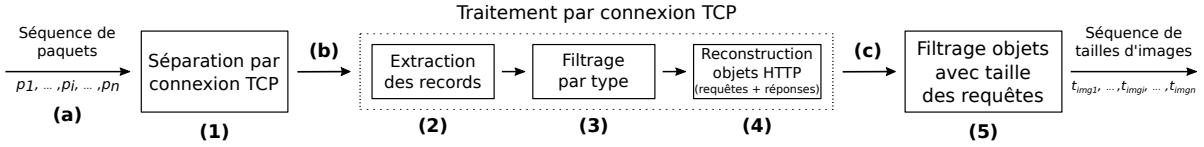


FIGURE 3.3 – Schématisation des étapes de la phase d'extraction des caractéristiques

Pour chaque trace réseau, nous récupérons (a) une séquence de paquets réseau. Nous traitons la séquence de paquets en commençant par séparer les connexions TCP (1).

Une connexion TCP est définie par une paire d'adresses IP (source et destination) et une paire de ports (port source client aléatoire et port 443 pour les services HTTPS). Il en résulte plusieurs séquences de paquets à traiter individuellement (b). Pour une séquence, nous pouvons accéder (2) aux records TLS et donc aux entêtes du protocole.

En filtrant les records TLS à l'aide du champ *type*, nous gardons uniquement les records liés aux données (*type* = APPLICATION_DATA) (3).

Pour chacun d'entre eux, nous pouvons extraire dans l'entête le champ indiquant la longueur et, en observant l'alternance entre les records en provenance ou à destination du serveur (pour du trafic HTTP/1.1), nous pouvons reconstruire la séquence des tailles des objets HTTP chiffrés envoyés par le serveur, ainsi que les requêtes associées envoyées par le client (4).

Suite à ce traitement, nous collectons une liste de tailles d'objets HTTP avec la longueur de leur requête associée (c). Grâce à ces informations, nous pouvons filtrer les objets envoyés par le serveur pour garder uniquement les images en utilisant comme critère les tailles des requêtes associées (5). En effet, nous avons remarqué que, pour les différents services étudiés (Google Images, DuckDuckGo Images, Instagram ou Amazon), la récupération des images lors du chargement des différentes pages s'opérait de façon similaire. Sur ces services, la page HTML fait des liens vers des images qui sont chargées dans un deuxième temps. Ces images sont hébergées sur un autre service et ces dernières images sont atteintes via une URL composée à chaque fois d'un chemin suivi d'un identifiant correspondant à l'image demandée. Comme l'ensemble des requêtes

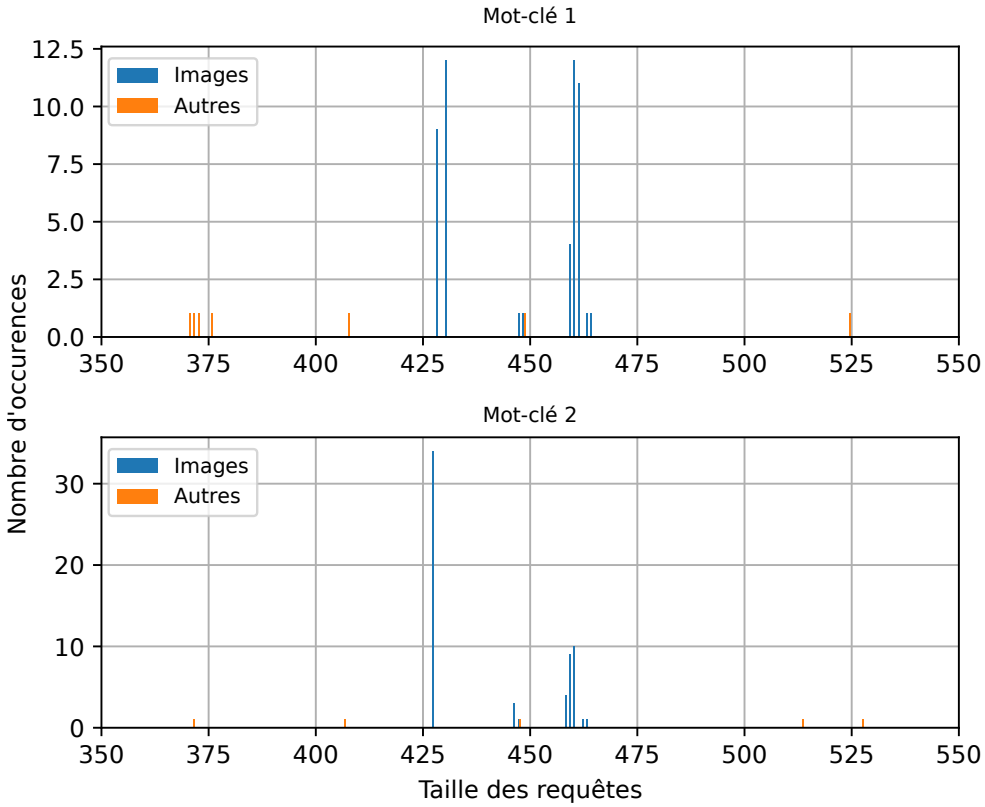


FIGURE 3.4 – Exemples de distribution de la tailles des requêtes pour la recherche de deux mots-clés différents sur le service Google Images

pour les images est similaire pour un service donné, il est possible de les distinguer des autres requêtes.

Par exemple, pour le service Google Images, les distributions des tailles des requêtes pour deux mots-clés sont tracées sur la figure 3.4. En bleu se trouvent les requêtes liées à des images et en orange les autres. On constate aisément que les requêtes concernant les images sont majoritaires et concentrées sur quelques tailles précises. Pour les distinguer nous avons deux options :

— (1) soit recenser toutes les tailles de requêtes connues qui peuvent renvoyer des images et reconnaître ces mêmes tailles lors de la phase de test pour effectuer un filtrage par liste blanche ;

— (2) soit appliquer un filtre qui ne garde que les tailles des requêtes où se concentre le plus grand nombre d'occurrences.

Pour le service Google Images, que nous étudions plus particulièrement dans la suite, nous avons utilisé la première option pour sa simplicité car l'ensemble des tailles des requêtes associé à des images était réduit (une vingtaine de tailles). Le filtrage peut également être appliqué à d'autres services à condition de développer un filtrage spécifique pour les différents services.

Pour résumer, pour toutes les traces réseau HTTPS (HTTP/1.1) nous collectons : la taille de la requête HTTP chiffrée, ainsi que la taille de l'objet HTTP chiffré correspondant à cette requête. Ensuite, à partir de la taille correspondant à la requête, nous filtrons les objets HTTP pour ne garder que ceux liés à des images.

3.3.3 Génération des signatures

Suite à l'étape d'extraction des caractéristiques, nous récupérons pour chaque mot-clé plusieurs ensembles de tailles d'images, correspondant chacun à une trace différente. Bien que le contenu de la page soit similaire, voire identique, pour le même mot-clé, nous observons empiriquement une variation mineure (quelques octets) de la taille des objets chiffrés. Ces différences s'expliquent par plusieurs facteurs.

Faible variabilité des tailles

Premièrement, bien que les images soient identiques, il y a des éléments qui peuvent introduire de la variabilité. En effet, lors du chargement de la même image pendant différentes itérations, la taille de l'image reste identique. Cependant, il nous faut revenir sur la notion d'objet HTTP. Cet objet est l'agrégation de l'entête HTTP de la réponse avec la réponse (payload) à proprement parler (cf. section 1.4.3 pour le fonctionnement des entêtes HTTP). Ainsi, même si le contenu est strictement identique, on rencontre dans les entêtes des informations comme l'âge du cache ou la date des données demandées. Ces informations évoluent au cours du temps et amènent ainsi de la variabilité.

Deuxièmement, l'objet est chiffré avant d'être envoyé. Lors du chiffrement d'un élément par un mode opératoire par bloc, ce dernier peut introduire du padding. En effet, il faut que la taille de l'ensemble des données à chiffrer soit un multiple de la taille des blocs. Ces blocs ont une taille de 128 bits, ou 16 octets. Ainsi, si on observe des petites variations au niveau des objets HTTP, cela peut avoir une répercussion au niveau de la taille après chiffrement par tranche de 16 octets.

Pour résoudre ce problème, nous allons faire appel aux techniques de classification supervisée, en particulier au partitionnement (clustering), afin de définir automatiquement des intervalles qui correspondraient à chaque image.

Classification à une dimension

Il existe de nombreuses solutions de partitionnement (clustering). Cependant, nous sommes dans un cas où nous avons uniquement une seule caractéristique en entrée. Ainsi, notre objectif est de partitionner un espace à une dimension. Pour ce type de problème, une des approches est de calculer la densité sur l'ensemble de l'espace possible et ainsi définir des intervalles où la densité est plus ou moins forte en fonction des probabilités d'usage des différentes valeurs.

Pour répondre à ce problème en particulier, nous avons retenu deux solutions : le modèle de mélange Gaussien et l'estimation par noyau. En effet, ces deux algorithmes permettent de générer la fonction de densité correspondant à une distribution aléatoire de données. Ces deux solutions sont présentées ci-dessous.

Le modèle de mélange Gaussien

Le modèle de mélange Gaussien, ou GMM pour Gaussian Mixture Model en anglais, est une méthode qui permet d'estimer la loi de probabilité d'une ou plusieurs variables aléatoires. Dans notre cas, nous nous intéressons à un problème à une dimension, donc avec une variable aléatoire. GMM nous permet d'exprimer une variable aléatoire comme une somme de plusieurs composantes gaussiennes.

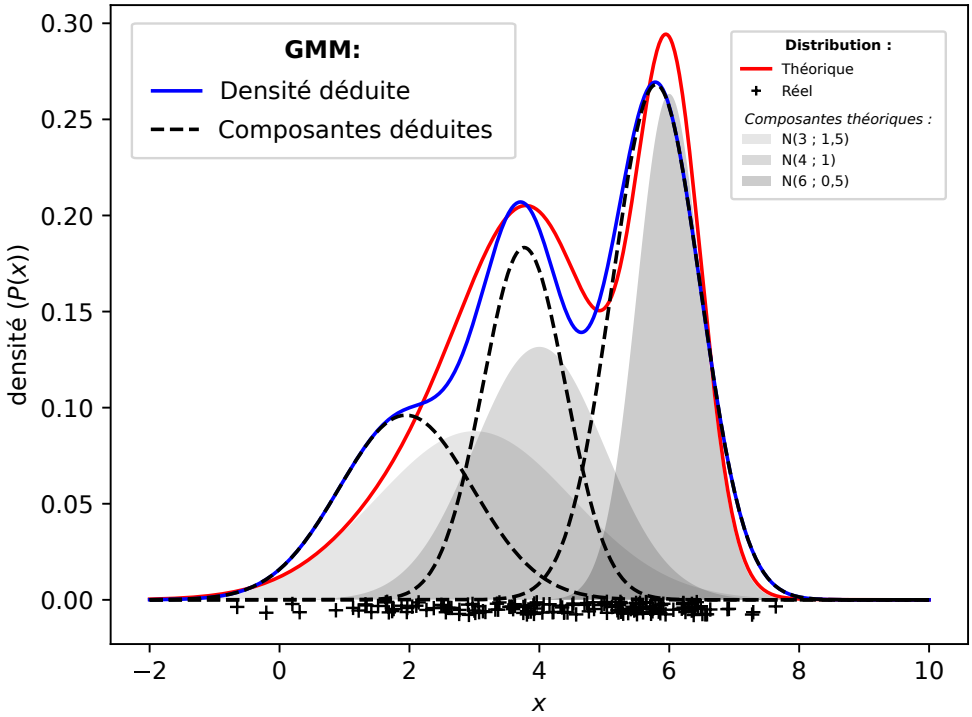


FIGURE 3.5 – Exemple d'application de GMM avec 3 composantes

La fonction de densité associée peut s'écrire :

$$g = \begin{cases} \mathbb{R} & \to \mathbb{R}_+ \\ y & \mapsto \sum_{i=1}^{N} \pi_i \mathcal{N}(\mu, \sigma^2)(y) \end{cases} \tag{3.9}$$

Avec N le nombre de composantes Gaussiennes définies à l'avance, π_i l'ensemble des poids de chaque composante, $\mathcal{N}$ une composante gaussienne de moyenne μ et de variance σ^2.

Les paramètres π_i, μ et σ^2 sont déterminés pour chaque composante lors de l'estimation des paramètres, qui utilise l'algorithme espérance-maximisation (EM) [109].

Sur la figure 3.5 nous avons tracé un exemple d'utilisation de GMM. Nous avons symbolisé avec des croix noires en bas du graphique les valeurs qui composent la distribution de test. Cette distribution est liée à la distribution théorique tracée en rouge qui est la somme des trois composantes Gaussienne en nuance de gris.

En appliquant GMM, avec trois composantes Gaussiennes sur la distribution réelle, nous obtenons la densité déduite représentée par la courbe bleu. Cette courbe bleu étant la somme des trois composantes déduites par GMM tracées en pointillées.

L'estimation par noyau

L'estimation par noyau, ou KDE pour Kernel Density Estimation en anglais, a été proposée pour la première fois en 1962 par Parzen E. [110]. Cette méthode permet d'estimer la fonction de densité d'une variable aléatoire.

Soit $X = (x_1, ..., x_n)$ une distribution aléatoire, K une fonction kernel et un paramètre de

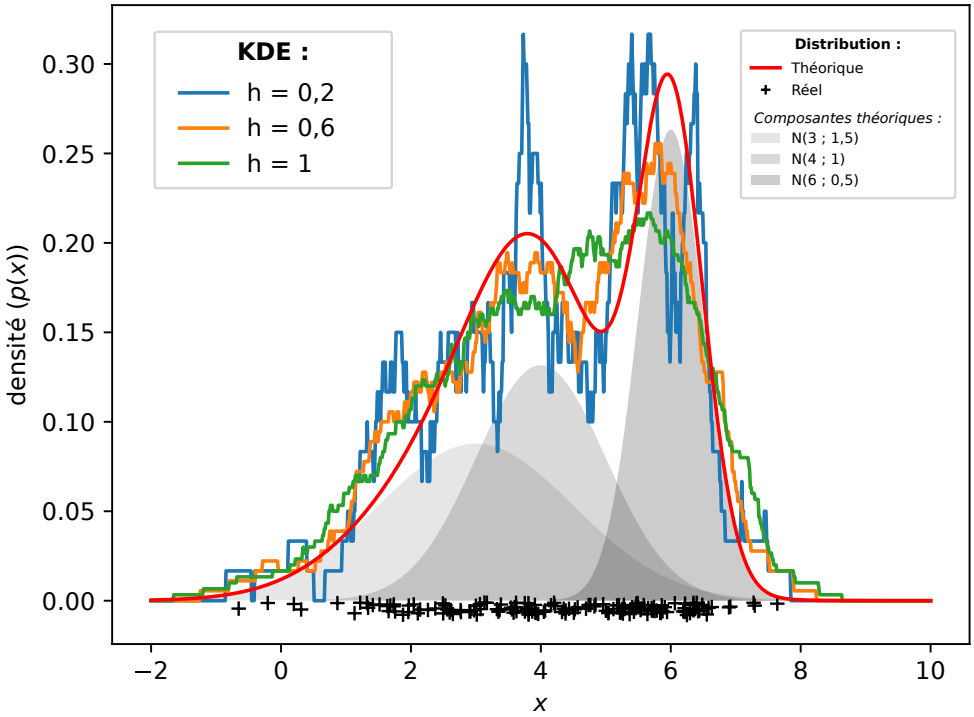


FIGURE 3.6 – Exemple d'application de KDE avec différents paramètres h

lissage h, la fonction de densité est définie comme :

$$f = \begin{cases} \mathbb{R} & \to \mathbb{R}_+ \\ y & \mapsto \frac{1}{nh} \sum_{i=1}^{n} K(\frac{y-x_i}{h}) \end{cases} \tag{3.10}$$

Le paramètre de lissage h fait varier la taille de la fenêtre de valeur prise en compte pour le lissage. De plus, le lissage de la courbe est également dépendant d'une fonction noyau (kernel). Cette dernière fait varier la forme du lissage et doit remplir quelques prérequis :

Définition	$K : \mathbb{R} \to \mathbb{R}_+$
Normalisation	$\int_{-\infty}^{+\infty} K(y)\,\mathrm{d}y = 1$
Symétrie	$\forall y \in \mathbb{R} : K(-y) = K(y)$

Sur la figure 3.6 nous avons tracé une exemple d'utilisation de KDE en considérant la même distribution théorique que pour GMM dans la figure 3.5. Pour rappel la courbe rouge est la distribution théorique qui est la somme de trois gaussienne (en nuance de gris). La distribution réelle est représentée par les croix en bas du graphique. Nous avons appliqué KDE avec une fonction noyau uniforme. Sur le graphique, nous avons tracé la densité calculée par KDE pour trois valeurs différentes de h le paramètre de lissage. Cela permet de constater qu'il est important de bien définir ce paramètre car s'il est trop faible la densité est bruitée et dans le cas contraire on perd en sensibilité et on perdra de l'information.

Comparaison des solutions

En termes de résultats, les expériences préliminaires que nous avons menées ne nous ont pas permis de statuer sur la supériorité d'une solution sur une autre. De fait, afin de les comparer,

nous considérons deux autres critères principaux : les paramètres des algorithmes et leurs performances.

Paramètres :

La première solution (KDE) requiert deux éléments : une fonction noyau K et un paramètre de lissage (ou largeur de fenêtre) h. Le premier paramètre n'est pas aussi influent que le deuxième et se trouve être souvent une densité de loi statistique (cf. 3.3.3). Le choix de ce paramètre s'effectue au regard des données considérées. Le deuxième paramètre va permettre de lisser la courbe en considérant plus ou moins de valeurs adjacentes. Plus le paramètre h sera élevé, plus la fenêtre de valeurs prises en compte sera grande également. Ce paramètre est essentiel pour les résultats de KDE, sa valeur s'évalue avec une partie des traces d'entraînement.

Le modèle de mélange Gaussien nécessite un paramètre d'entrée : le nombre de composantes. Cette solution nécessite, dans notre cas, de connaître le nombre d'images par trace (une image = une composante). Cependant, il est important de noter que, dans notre jeu de données, le nombre d'images n'est pas nécessairement constant d'une trace à une autre pour un même mot-clé. Cela provient soit du filtrage des objets HTTP, soit du chargement de la page. On peut cependant considérer le nombre moyen d'images. GMM nécessite ensuite de fixer l'ensemble des poids et les paramètres de chaque composante (moyenne et variance). Ces éléments sont sélectionnés automatiquement à partir des données d'entraînement grâce à l'algorithme EM.

Temps de calcul :

Afin de déterminer la solution la plus efficace au niveau du temps de calcul, nous avons construit une expérience pour les comparer. Nous avons enregistré le temps de calcul moyen sur 10 expériences pour les différents cas de figure suivants : temps de calcul pour l'apprentissage d'une signature, temps de calcul pour le test d'une trace face à une signature. Nous avons fait varier le nombre d'images qui composent la signature pour voir l'évolution des temps de calcul en fonction du nombre de tailles d'images dans une signature pour la phase d'apprentissage et de test.

GMM est paramétré avec 60 composantes car les traces du mot-clé considéré ont en moyenne 60 images. Pour KDE, nous utilisons le kernel uniforme et une valeur de lissage $h = 16$ (valeur optimale dans notre cas, cf. paragraphe 4.3.2).

Les résultats sont présentés sur les graphiques de la figure 3.7. On voit nettement que les temps de calcul sont plus faibles pour KDE que pour les GMM, que ce soit pour la phase d'apprentissage ou la phase de test.

Pour la phase d'apprentissage, les résultats sont cohérents car KDE n'a pas de réelle phase d'apprentissage alors que GMM doit quant à lui fixer les différents paramètres π, μ et σ^2 avec l'algorithme EM.

Pour la phase de test, on s'attend à ce que GMM soit plus rapide. En effet, la phase de test pour une signature de GMM s'opère en temps constant par rapport au nombre de tailles composant la signature. Le test consiste à évaluer chaque composante pour chaque taille de la trace testée (voir section 3.3.4). Si on note n le nombre de tailles composant la signature et m le nombre de tailles composant la trace de test, GMM a une complexité en $O(m)$. Pour KDE, la phase de test dépend de n et de m et a donc une complexité en $O(mn)$. Cependant, les résultats de la figure 3.7 montrent le contraire. Cela s'explique par le faible nombre d'éléments qui

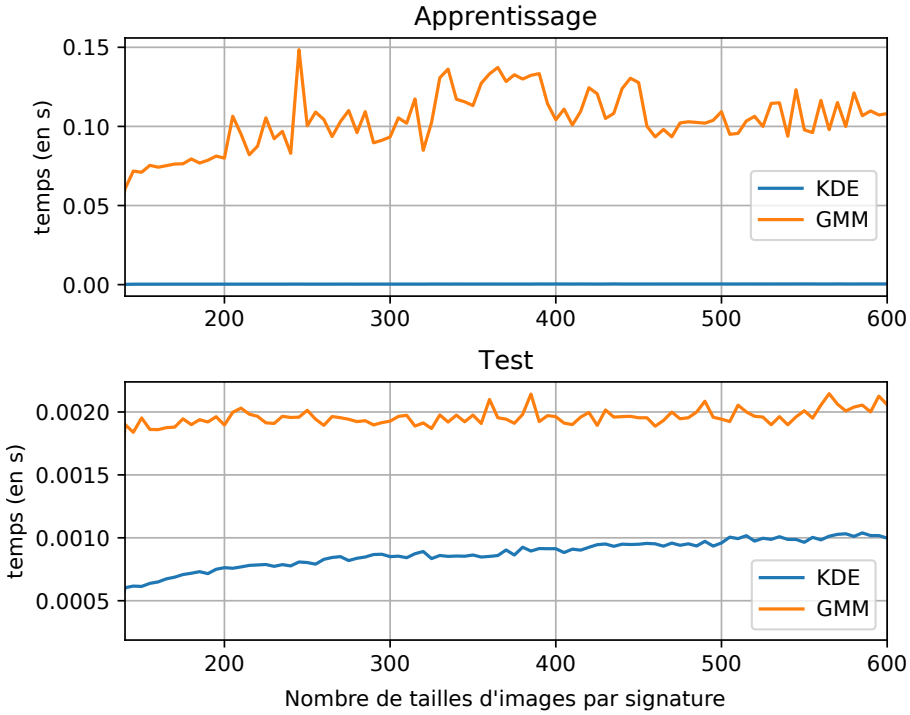


FIGURE 3.7 – Comparaison du temps de calcul (apprentissage et test) pour les algorithmes KDE et GMM

composent nos signatures : en moyenne 200 éléments (50 images en moyenne × 4 traces). Sur la figure 3.8, nous avons tracé les courbes pour une signature composée de plus de tailles d'images (entre 100 et 30 000). On observe donc qu'il faut considérer plus d'éléments par signature pour que l'usage de GMM soit plus intéressant (environ 4000).

Finalement, d'un point de vue temps de calcul, le choix se porte sur KDE pour notre problème. De plus, KDE ne nécessite pas de définir le nombre de tailles d'images par signature contrairement à GMM et comme nous l'avons vu cette valeur peut varier d'une trace à l'autre. Ainsi, éviter ce paramètre permet d'éviter des erreurs dues à ces variations. KDE nécessite cependant plus de paramètres, mais ces derniers sont à fixer globalement et non pas par signature. Pour ces différentes raisons, dans la suite de nos travaux, nous utiliserons la méthode de l'estimation par noyau (KDE).

Fonction signature

Soit une distribution de tailles d'images chiffrées pour un mot-clé ; KDE nous permet d'estimer la fonction de densité de cette distribution. Nous sommes alors capables de créer une fonction qui agrège les tailles récupérées dans un ensemble de traces pour créer notre signature. L'agrégation nous permet de considérer les faibles variations observées pour une image au travers des traces d'entraînement. Ainsi, KDE permet de lisser la distribution sur l'ensemble des traces.

Par conséquent, la fonction que nous dérivons représente la signature d'un mot-clé et, par la suite, nous pourrons tester si un ensemble de tailles d'images (issue d'une trace) correspond à cette signature ou non.

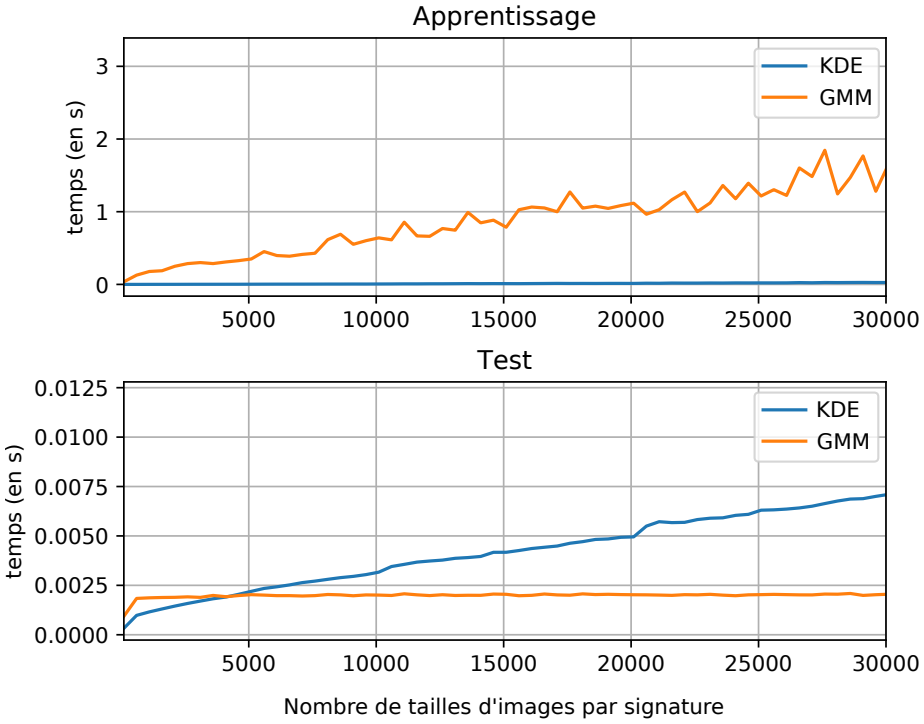


FIGURE 3.8 – Comparaison du temps de calcul (apprentissage et test) pour les algorithmes KDE et GMM avec un nombre de tailles par signature élevé

Formellement, dans notre contexte, une distribution est composée de toutes les n tailles d'images chiffrées t_i pour un mot-clé m_{nl}. Pour cette raison nous utilisons le noyau uniforme afin de garder un noyau généraliste.

$$K(y) = \frac{1}{2} \text{ si } \mid y \mid \leq 1 \text{ sinon } 0 \tag{3.11}$$

En considérant les équations (3.10) et (3.11), la fonction signature, notée σ_{mnl}, est définie à partir des tailles des images chiffrées t_i :

$$\sigma_{mnl}(y) = \frac{1}{nh} \sum_{i=1}^{n} (\frac{1}{2} \text{ si } \mid y - t_i \mid \leq h \text{ sinon } 0) \tag{3.12}$$

$$\sigma_{m}(x) = \frac{1}{nh} \sum_{i=1}^{n} (\frac{1}{2} \text{ si } \mid x - t_i \mid \leq h \text{ sinon } 0) \tag{3.13}$$

Finalement, notre base de données de signatures D est composée de toutes les fonctions de densité associées aux différents mots-clés et peut s'écrire : $D = \{\sigma_{mnl}, m_{nl} \in M_{nl}\}$

3.3.4 Algorithme de classification

D'après l'équation (3.8), la classification prend en entrée une liste de tailles d'images et retourne un mot-clé m_{nl} de M_{nl} ou leg (pour légitime) si la trace ne correspond pas à un mot-clé

que l'on recherche. Le processus complet est séparé en deux étapes : évaluation des signatures et filtrage des scores. Un schéma récapitulatif est disponible sur la figure 3.9.

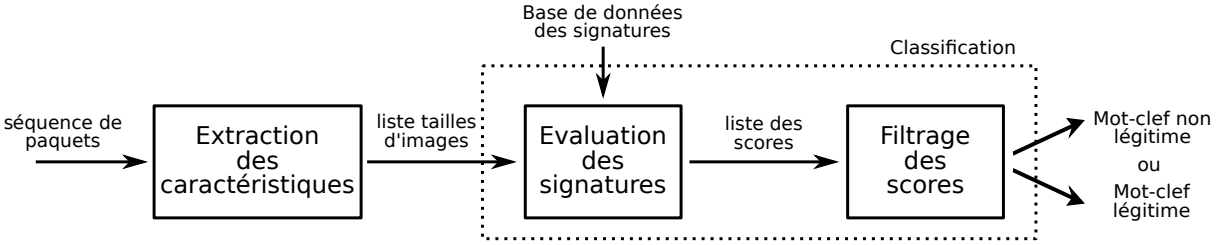


FIGURE 3.9 – Positionnement et composition de l'étape de classification dans la phase de supervision

Avant la phase de classification, les tailles des images contenues dans les paquets doivent être extraites. Cette extraction est réalisée de la même manière que pendant l'apprentissage (cf. Section 3.3.2). En sortie de cette étape, nous avons une séquence de tailles d'images chiffrées.

On note cette liste : $l_t = \{t_1, \ldots, t_i, \ldots, t_n\}$.

Évaluation des signatures

La liste l_t est évaluée en considérant les différentes signatures de la base de données D (définie au préalable). Les tests permettent d'identifier la signature correspondant au mieux à la trace en cours d'analyse. L'évaluation des signatures (fonctions de densité) consiste à calculer un score pour chaque signature de D correspondant à cette trace. Ce score se calcule en évaluant la fonction signature en chaque point de la trace (tailles) et en sommant le score obtenu en chaque point. Formellement, nous pouvons le définir comme suit :

Pour chaque signature $\sigma_{m_{nl}} \in D$, le score correspondant se note $sc_{m_{nl}}$:

$$sc_{m_{nl}}(l_t) = \sum_{i=1}^{n} \sigma_{m_{nl}}(t_i) \tag{3.14}$$

Après avoir calculé le score pour chaque signature, le mot-clé correspondant à la signature avec le score le plus élevé est considéré comme le plus probable.

Nous notons ce mot-clé m_{max} :

$$m_{max} = \underset{m' \in M_{nl}}{\arg\max}\, sc_{m'}(l_t) \tag{3.15}$$

Cependant, comme notre classification a pour but d'être applicable dans un contexte contenant des mots légitimes (monde ouvert), il nous faut une étape supplémentaire pour vérifier ce cas-là.

Filtrer les résultats

Comme il n'y a pas de signature pour les mots-clés légitimes, il nous faut une solution pour les identifier à partir des résultats obtenus avec les signatures de D. Le filtrage repose sur un score de confiance par rapport au score moyen pour toutes les signatures et par rapport au score maximal obtenu pour le meilleur mot-clé.

Le score maximal de la liste des scores est considéré comme pertinent uniquement dans le cas où il est supérieur à un certain seuil, sinon le mot-clé est considéré comme légitime. Ce seuil est défini à partir de la moyenne (noté $\bar{x}$), de l'écart-type (noté ϵ) de la liste des scores et d'un paramètre du filtrage α. Il s'agit d'une constante que nous réglerons expérimentalement pour optimiser les résultat de la classification.

Formellement, soit $l_{sc} = \{sc_{m_{nl_1}}(l_t), ..., sc_{m_{nl_i}}(l_t), ..., sc_{m_{nl_n}}(l_t)\}$ la liste des scores obtenues à l'étape précédente, le filtrage se définit de la manière suivante :

$$f(l_t) = \begin{cases} m_{max} \text{ si } sc_{m_{max}}(l_t) > \bar{x}(l_{sc}) + \alpha \times \epsilon(l_{sc}) \\ leg\ sinon \end{cases} \tag{3.16}$$

Autrement dit, le paramètre de filtrage α multiplié par l'écart-type définit la valeur que doit atteindre le score par rapport à la moyenne pour être considéré comme lié à un mot-clé non légitime. En faisant varier le paramètre α, l'administrateur peut ajuster la balance entre le taux de faux-positifs et le taux de faux-négatifs.

3.4 Synthèse

Suite à l'élaboration d'une modélisation du trafic chiffré spécifique au protocole HTTPS couplé avec la version 1.1 de HTTP, nous avons pu remarquer qu'il était possible de reconstruire la taille des objets HTTP chiffré transitant sur le réseau. Nous avons montré qu'il était possible d'extraire de l'ensemble des objets HTTP, ceux qui correspondaient à des images. Ensuite nous avons démontré théoriquement qu'il était possible de caractériser le trafic issu du chargement de la page associée à un mot-clé sur un service de recherche d'image. La caractérisation du trafic nous a permis de mettre au point une méthode pour superviser l'usage de certains mots-clés sur des services spécifiques.

Nous considérons pour cette méthode un scénario en monde ouvert où le but est de reconnaître l'usage de mots-clés non légitimes parmi des mots-clés légitimes et pour ceux non légitimes, de les identifier spécifiquement. Dans ce scénario nous ne connaissons pas les mots-clés légitimes, ce qui garantit le respect de la vie privée des utilisateurs légitimes.

La méthode que nous avons mise en place se construit autour de deux phases. La première correspond à la génération des signatures pour les mots-clés non légitimes que nous souhaitons reconnaître. Cette phase se découpe en plusieurs étapes : la collecte des traces d'entraînement, l'extraction des caractéristiques et l'apprentissage des signatures. Nous extrayons du trafic la séquence des tailles du chiffré des images appartenant à une trace. Pour combiner les caractéristiques provenant de l'ensemble des traces d'entraînement et construire une signature nous utilisons l'estimation par noyau (KDE). Finalement nous constituons une base de données de signatures pour l'ensemble des mots-clés non légitimes.

Après la phase de construction des signatures, nous pouvons mettre en œuvre la phase de supervision. La supervision comprend la capture des traces à analyser, l'extraction des caractéristiques du trafic et enfin la classification. Nous extrayons les mêmes caractéristiques du trafic que lors de l'apprentissage mais cette fois nous utilisons ces informations pour les évaluer avec notre base de données de signatures. Une fois l'ensemble des signatures analysées nous désignons la signature la plus proche et ainsi nous reconnaissons le mot-clé associé. Cependant, avant de

considérer cette réponse comme définitive, la classification effectue une dernière analyse statistique afin de reconnaître si la trace est légitime ou non.

Dans le chapitre suivant, nous mettons en pratique la méthode de classification décrite dans ce chapitre via notre implémentation nommée H1Classifier et nous l'évaluons.

Chapitre 4

Évaluation de la classification : H1Classifier

Sommaire

4.1 Introduction 62
4.2 Présentation du jeu de données 62
4.2.1 Génération des traces 62
4.2.2 Jeu de données 63
4.3 Expérimentation et résultats 64
4.3.1 Métriques pour l'évaluation 65
4.3.2 Évaluation des paramètres 67
4.3.3 Expérience à grande échelle 69
4.3.4 Discussion 72
4.4 Classifier du trafic HTTP/2 avec H1Classifier 73
4.4.1 Recherche d'une nouvelle caractéristique 74
4.4.2 Évaluation de H1Classifier avec du trafic HTTP/2 75
4.4.3 Limitations du portage de la méthode 76
4.4.4 Test du caractère discriminant des signatures 77
4.5 Synthèse 79

4.1 Introduction

L'objectif principal de ce chapitre est d'évaluer l'approche que nous avons présentée dans le chapitre 3. Pour cela, nous avons développé un outil complet, appelé H1Classifier qui intègre à la fois la génération des signatures et la supervision.

Concernant la génération des signature, il a fallu dans un premier temps obtenir des données et nous nous sommes ainsi interrogés sur comment construire un jeu de données représentatif. Nous commençons, ainsi, ce chapitre par la présentation du jeu de données que nous avons construit. Cet ensemble de traces sert à l'apprentissage des signatures pour les mots-clés non légitimes que nous souhaitons détecter. Le jeu de données contient également d'autres traces qui assurent le rôle de traces de tests pour la phase d'évaluation. Nous présentons également notre robot de collecte de traces.

Pour la partie supervision, nous étudions l'impact des paramètres de H1Classifier afin de faciliter la mise en œuvre de la solution. Nous définissons un ensemble de métriques pour une évaluation en monde ouvert, multi-niveaux, puis nous évaluons H1Classifier selon plusieurs objectifs :

— Étudier son comportement avec un grand nombre de mots-clés (légitimes et non légitimes).

— Comprendre l'évolution du taux de justesse et de faux-positifs en fonction du nombre de mots-clés supervisés ou du pourcentage de traces légitimes.

— Tester son applicabilité avec du trafic HTTP/2.

4.2 Présentation du jeu de données

Le jeu de données pour l'expérimentation sur H1Classifier se focalise sur le service Google Images. En effet, comme le montre le paragraphe 3.2.2, Google Images est le service avec la variabilité la plus faible du point de vue des tailles d'images (cf. figure 3.1). Ainsi, si la solution fonctionne pour ce service, a priori cela doit fonctionner sur les autres services où la taille des images est plus variable et donc plus discriminante. Pour l'expérimentation, notre but est de détecter 10 500 mots-clés considérés comme non légitimes. Le trafic légitime est, quant à lui, représenté par 105 000 autres mots-clés. Nous avons fait le choix d'avoir un ratio d'un dixième entre les traces légitimes et non légitimes. Ce ratio nous permet d'avoir une quantité suffisante de mots-clés non légitimes tout en gardant une majorité de mots-clés légitimes afin d'être plus fidèle à la réalité.

4.2.1 Génération des traces

Afin de générer les traces nécessaires à l'apprentissage des signatures et à la phase d'évaluation, nous avons développé un collecteur automatique de traces réseau. Notre collecteur accède à des URLs et interagit avec la page au moyen de commandes javascript. Dans notre cas, nous nous intéressons à identifier l'usage de mots-clés sur le service de Google Images. Nous pouvons accéder directement aux différentes pages en passant par une URL dont le modèle est le suivant : `https://www.google.com/search?tbm=isch&q=<mot-cle>` ; le paramètre "tbm=isch" correspond à la section Images de Google et un deuxième paramètre "q=<mot-clé>" contient le mot-clé demandé.

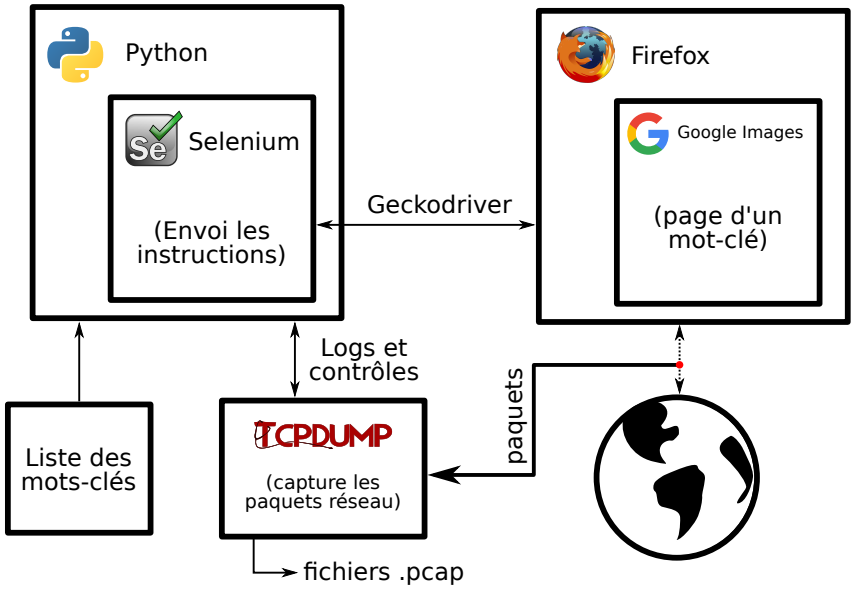


FIGURE 4.1 – Schéma outil de capture

Le collecteur est implémenté en Python et utilise la bibliothèque Selenium [34] couplée au logiciel geckodriver [35]. Selenium est un logiciel de test pour les navigateurs web, il permet également d'interagir avec ces derniers, et donc de simuler le comportement d'un utilisateur de manière automatisée. Geckodriver est un pilote logiciel qui fait le lien entre Selenium et le navigateur Firefox. Avec ces outils, nous avons donc développé une solution permettant de visiter automatiquement une liste de mots-clés prédéfinis sur le service Google Images.

Parallèlement à l'accès aux différentes pages, notre solution capture tous les paquets réseau entrant et sortant de la machine. Pour cela, nous utilisons le logiciel tcpdump [36], qui embarque la librairie de référence LibPcap pour la capture réseau. Afin de pouvoir identifier aisément le début et la fin de chaque trace réseau, nous gardons dans des logs les horodatages de début et de fin de chargement de chaque page, ainsi que le mot-clé correspondant. Ainsi, nous récupérons une ou plusieurs séquences de paquets réseau pour chaque mot-clé. La figure 4.1 schématise et positionne les différents outils qui communiquent entre eux.

4.2.2 Jeu de données

Pour avoir une évaluation réaliste, nous considérons une situation dite en monde ouvert. Cela signifie que les traces considérées pendant l'entraînement ne couvrent pas la totalité de l'espace des possibles. Par conséquent, il faut une solution qui appréhende ce que l'on ne connaît pas encore. Dans notre cas précis, nous apprenons des mots-clés non légitimes mais nous n'avons pas connaissance de l'ensemble des mots-clés légitimes, ce qui garantit le respect de la vie privée. Comme nous l'avons indiqué dans l'introduction de cette section, nous engendrons dix fois plus de trafic légitime que de trafic non légitime dans la phase de test.

34. `https://www.seleniumhq.org/`
35. `https://github.com/mozilla/geckodriver`
36. `https://www.tcpdump.org/`

TABLE 4.1 – Jeu de données pour le trafic HTTP/1.1

ID	Origine des mots-clés	Nombre de mots-clés	Captures par mots-clés	Nombre moyen de paquets par trace	Durée moyenne d'une trace
$data_1$	Dictionnaire Anglais	10 500	5	1155 pqt/trace	2,94 s/trace
$data_2$	Titre de pages Wikipedia (en)	105 000	1		

Les mots-clés assimilés comme non légitimes pour notre expérience sont extraits du dictionnaire anglais : le *Oxford Dictionary*[37]. Pour la suite du document, nous nommons l'ensemble des traces liées à ces mots-clés : $data_1$. $data_1$ est composé de traces réseau de 10 500 mots-clés différents, et chaque mot-clé possède cinq traces réseau différentes, correspondant au chargement de la page d'un mot-clé.

La deuxième partie des mots-clés correspond aux requêtes légitimes et est regroupée dans un ensemble de traces nommées : $data_2$. $data_2$ est composé de 105 000 traces qui sont chacune liée à un mot-clé unique. De plus, tous les mots-clés de $data_2$ sont exclus de $data_1$. Les mots-clés sont des mots ou expressions extraits des titres de pages d'articles de Wikipédia. La liste complète des titres est accessible à l'adresse suivante : `dumps.wikimedia.org/enwiki/latest/enwiki-latest-all-titles-in-ns0.gz`. Nous avons utilisé une deuxième source de mots-clés car le dictionnaire en libre accès est limitée et de taille insuffisante.

Le tableau 4.1 contient un résumé des informations sur le jeu de données. Toutes les traces réseau pour ce jeu de données, que ce soit $data_1$ ou $data_2$, ont été capturées pendant les mois de juin et juillet 2017. Le jeu de données a été entièrement généré depuis une machine dédiée, avec la configuration suivante :

— OS : Debian 8
— Navigateur : Firefox 54.0.1 (64-bit)
— Options navigateur : cache désactivé, HTTP/2 désactivé afin de forcer HTTP/1.1
— Version TLS : 1.2
— Résolution écran : 1920×1080, navigateur en fenêtre agrandie
— Google Images : version anglaise (`www.google.com`) obtenue par désactivation de la redirection locale[38].

4.3 Expérimentation et résultats

H1Classifier est l'implémentation de notre méthode. Nous l'avons développé en utilisant le langage python et nous utilisons la librairie scikit-learn [111] pour la partie KDE. Cette section présente les expériences que nous avons menées pour évaluer H1Classifier, ainsi que les résultats obtenus. Pour cela, nous définissons tout d'abord les métriques d'évaluation. Nous présentons ensuite nos expérimentations, qui se décomposent en deux parties. La première expérience évalue

37. `https://raw.githubusercontent.com/sujithps/Dictionary/master/Oxford%20English%20Dictionary.txt`
38. Se rendre sur `https://www.google.com/ncr` pour désactiver la redirection automatique.

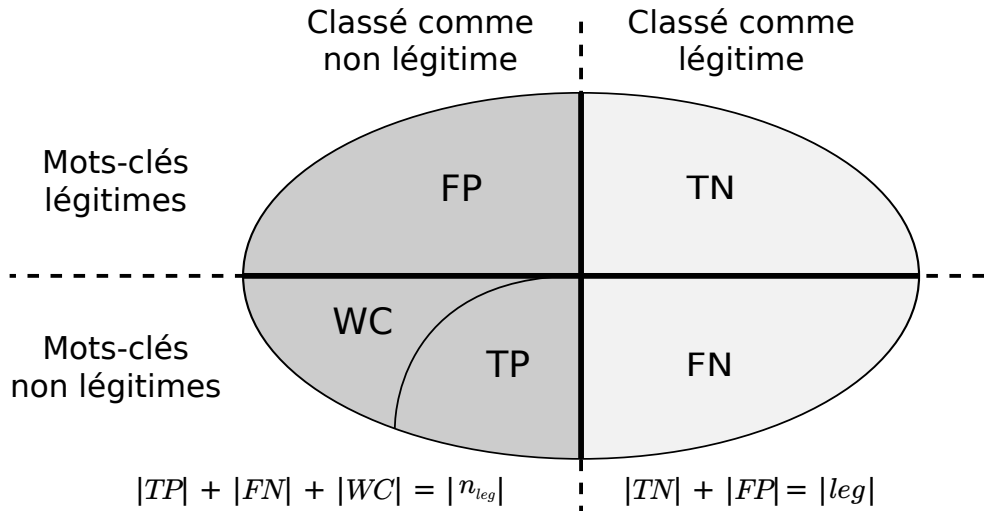


FIGURE 4.2 – Définition des métriques

les paramètres de notre méthode afin de trouver leurs valeurs optimales et comprendre leur impact sur l'efficacité de notre technique. Nous utilisons pour cette expérience une partie réduite du jeu de données en nous limitant à $data_1$. La deuxième expérience considère l'ensemble des données collectées ($data_1$ et $data_2$) avec les paramètres définis précédemment. Ces derniers résultats nous permettent d'évaluer notre solution dans un cas étendu.

4.3.1 Métriques pour l'évaluation

En réutilisant les notation de l'équation (3.8), nous notons p_i une trace composée d'une séquence de paquets, et $m \in \{M_{nl} \cup \{leg\}\}$ son véritable mot-clé associé. $f(p_i)$ est le mot-clé prédit par la classification. Il est considéré comme :

TP (Vrai Positif ou True Positive) si $m \neq leg$ et $m = f(p_i)$. La trace est reconnue comme un mot-clé non légitime et est classée comme telle avec le bon mot-clé.

FP (Faux Positif ou False Positive) si $m = leg$ et $leg \neq f(p_i)$. Cela signifie que la trace correspond à un mot légitime mais qu'elle est classée comme non légitime par le système à tort.

TN (Vrai Négatif ou True Negative) si $m = leg$ et $leg = f(p_i)$. La trace est reconnue comme un mot-clé légitime à juste titre.

FN (Faux Négatif ou False Negative) si $m \neq leg$ et $leg = f(p_i)$. Cela signifie que la trace correspond à un mot clé non légitime mais qu'elle est classée comme étant légitime par le système à tort.

WC (Mauvaise Classification ou Wrong Classification) si $m \neq leg$ et $m \neq f(p_i)$. La trace p_i associée à un mot-clé de M_{nl} a été correctement classée comme non légitime mais le mot-clé prédit n'est pas correct.

Nous distinguons entre TP et WC car notre solution est testée à deux niveaux de granularité. Le premier consiste à uniquement détecter si une trace correspond à un mot-clé légitime ou non, sans faire de distinction précise entre les mots-clés. Dans ce cas-là, les traces arrivant dans

la catégorie WC ne sont pas considérées comme des erreurs. Mais dans le cas où le but est d'identifier précisément les mots-clés, il est important de dissocier les traces de la catégorie WC de celles qui sont des TP. Dans la suite, les métriques TN et FN ne seront pas utilisées car ces dernières sont redondantes par rapport à FP, WC et TP. En effet à partir des trois dernières métriques, les deux premières sont totalement définies comme on le voit avec les équations en bas de la figure 4.2.

Pour la suite, nous notons $|nleg|$ le nombre de traces correspondant à des termes non légitimes et $|leg|$ le nombre de traces légitimes. En considérant toutes les traces p_i : $|TP|$, $|FP|$ et $|WC|$ sont respectivement le nombre de traces classées comme TP, FP et WC. Ainsi, nous pouvons définir les métriques suivantes accompagnées de leur interprétation :

TPR (True Positive Rate ou taux de vrai positif) : $\frac{|TP|}{|nleg|}$. Il s'agit de la probabilité qu'une trace d'un mot-clé non légitime soit correctement classée comme telle avec le bon mot-clé. Plus cette valeur est élevée et plus la solution labellise correctement les traces non légitimes. A l'inverse si elle est faible cela signifie que la solution éprouve des difficultés à soit (1) reconnaître les traces non légitimes (sans distinction du mot-clé) soit (2) les labelliser (c'est à dire y apposer le bon mot-clé). Pour trancher ce cas, il est nécessaire que regarder les métrique WCR et FNR.

FNR (False Negative Rate ou taux de faux négatif) : $\frac{|FN|}{|nleg|}$. Il s'agit de la probabilité qu'une trace d'un mot-clé non légitime soit incorrectement classée comme légitime. Cette métrique indique, lorsqu'elle est élevée, que la solution éprouve des difficultés à reconnaître les traces légitimes : cas (1).

WCR (Wrong Classification Rate ou taux de mauvaise classification) : $\frac{|WC|}{|nleg|}$. Il s'agit de la probabilité qu'une trace d'un mot-clé non légitime soit classée comme un autre mot-clé non légitime. Cette métrique à l'inverse du FNR, permet d'évaluer si la solution labellise correctement une trace non légitime lorsqu'elle est élevée : cas (2). Il y a un lien très fort entre ces trois dernières métriques définies par cette égalité : $TPR + FNR + WCR = 1$.

FPR (False Positive Rate ou taux de faux positif) : $\frac{|FP|}{|leg|}$. Il s'agit de la probabilité qu'une trace d'un mot-clé légitime soit incorrectement classée comme non légitime. Plus cette valeur est élevée et plus les détections de mauvais comportements seront moins fiables. En effet plus le FPR est grand et plus il y a des traces légitimes qui sont reconnues comme non légitimes. Cette métrique doit rester la plus faible possible pour que l'on puisse avoir confiance dans les résultats.

TNR (True Negative Rate ou taux de vrai négatif) : $\frac{|TN|}{|leg|}$. Il s'agit de la probabilité qu'une trace d'un mot-clé légitime soit correctement classée comme telle. Cette métrique mesure la capacité de la solution à détecter les traces légitimes. Ce taux est le complément à un de FPR car $FPR + TNR = 1$.

Acc (Accuracy ou Justesse) : $\frac{|TP|+|TN|}{|nleg|+|leg|}$. Il s'agit du ratio de traces ayant été correctement classées, indépendamment de leurs natures (légitimes ou non). La justesse mesure la performance globale de la solution, plus cette valeur est élevée et moins il y a d'erreur de classification.

Un récapitulatif schématique des différentes métriques est présenté dans la figure 4.2. Nous portons l'attention du lecteur sur le fait qu'avec l'introduction du WCR : $FNR + TPR$ ne vaut pas systématiquement 1 (uniquement si le WCR est nul), contrairement aux métriques classiques

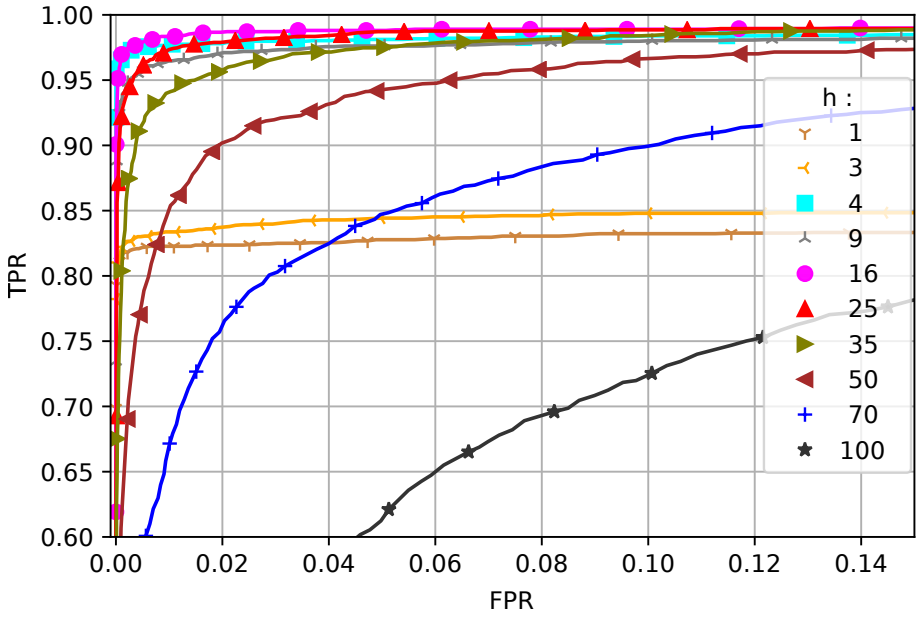


FIGURE 4.3 – Courbe ROC : évaluation de la largeur de fenêtre h ($data_1$)

de classification.

4.3.2 Évaluation des paramètres

Nous mesurons, ici, l'impact de la largeur de fenêtre (h) et du seuil de filtrage (α) sur les performances de la solution H1Classifier. En plus de ces paramètres, nous avons étudié l'influence du nombre de mots-clés non légitimes ($|M_{nl}|$) appris par notre solution. Pour ces expériences, nous utilisons uniquement le jeu de données $data_1$. Ce jeu de données est de taille suffisante car il ne s'agit pas ici de tester le passage à l'échelle.

Largeur de fenêtre h

Afin de déterminer les effets de la largeur de fenêtre, nous utilisons un dixième (1050) des mots-clés en tant que mots-clés non légitimes. Nous rappelons que h est un paramètre de KDE. Avec quatre des cinq traces disponibles pour ces mots-clés, nous avons construit leurs signatures respectives. Une des cinq traces de chacun des mots-clés non légitimes est mise de côté pour le test, ainsi qu'une trace pour chacun des 9450 mots-clés considérés comme légitimes. L'ensemble de ces traces constitue le jeu de test (9450 + 1050 traces). L'évaluation est reproduite trois fois de manière indépendante. Pour chacune de ces évaluations, les 1050 mots-clés non légitimes sont sélectionnés au hasard (avec remise). Il en est de même pour les traces retenues pour le jeu de test.

Les taux de faux positifs (FPR) et de vrais positifs (TPR) obtenus après l'évaluation sont indiqués sur les courbes ROC (Receiver Operation Characteristic, ou fonction d'efficacité du récepteur) de la figure 4.3, en variant le paramètre de filtrage α. On remarque que pour une fenêtre h comprise entre 1 et 16, les résultats semblent s'améliorer à l'exception de la valeur $h = 9$ où nous observons une légère dégradation. La classification atteint son meilleur résultat pour la valeur 16 où l'on observe moins de 0,2% de faux positifs pour plus de 96,8% de vrais

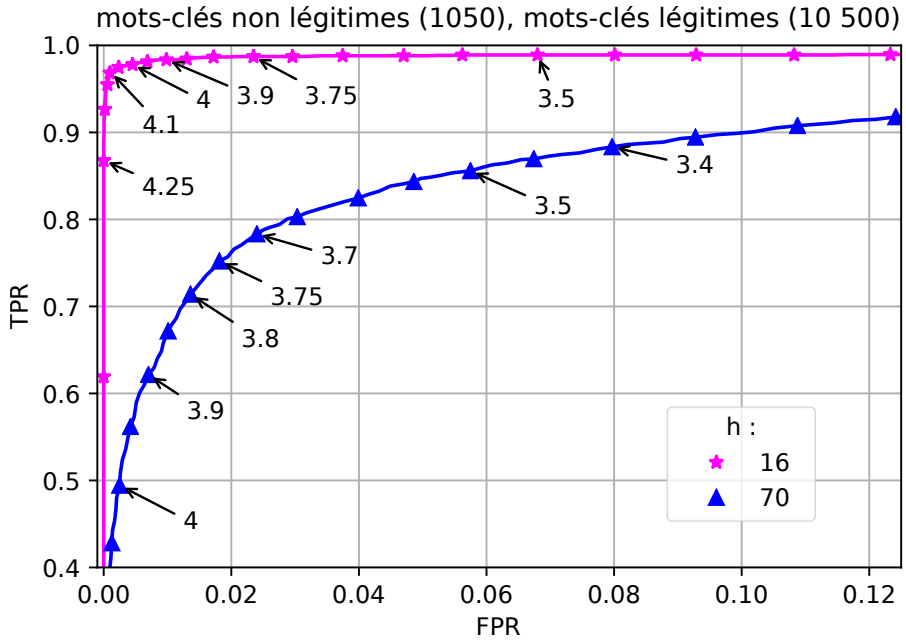


FIGURE 4.4 – Impact du paramètre fenêtre (h) sur le TPR et le FPR en faisant varier le paramètre α ($data_1$)

positifs.

Dans le détail, la capacité de détection croît entre les valeurs 1 et 4 de h. Entre 5 et 16, la performance commence par décroître, puis remonte de nouveau pour atteindre la valeur maximale pour 16. Pour les largeurs de fenêtres supérieures à 16, le TPR décroît au fur et à mesure que la largeur h augmente. Ces variations sont liées aux variations dans les tailles du chiffrées des images que nous récupérons, comme nous l'avons expliqué dans le paragraphe 3.3.3 (chiffrement par bloc). Expérimentalement, nous en déduisons que les variations des tailles sont majoritairement dans un intervalle de +/- 4 octets autour de la valeur moyenne, bien qu'il y en ait quelques autres dans l'intervalle +/- 16 octets. Nous considérons ainsi, que la valeur $h = 16$ est optimale.

Seuil de filtrage α

Pour rappel le paramètre α est le niveau de filtrage entre les trace légitimes et non légitimes. Ce filtrage permet de voir si le meilleur score obtenu ($sc_{m_{max}}$), pour un certain mot-clé (m_{max}), se démarque des autres scores. Plus α est grand et plus $sc_{m_{max}}$ doit être élevé par rapport à la moyenne de l'ensemble des scores afin que m_{max} soit conservé. Si ce n'est pas la cas la trace testée est classé en tant que trace légitime (leg).

Pour l'évaluation de ce paramètre, nous reprenons les courbes ROC de la figure 4.3 pour deux valeurs de fenêtre fixes : $h = 16$ et $h = 70$. Ces valeurs correspondent respectivement à la valeur optimale définie précédemment, et à une autre valeur avec des résultats inférieurs. Les deux courbes correspondantes sont présentées dans la figure 4.4 qui renseigne également les points correspondant aux variations du paramètre α. Nous observons que plus α est élevé, plus le TPR croit et plus le FPR diminue. Ce résultat est logique car plus le filtrage est sélectif, plus la solution classe les traces dans la partie légitime. Il est intéressant de noter que le seuil pour le paramètre α dépend également du paramètre h. En effet, la valeur du paramètre de lissage h a un impact direct sur le score moyen calculé avant l'opération de filtrage.

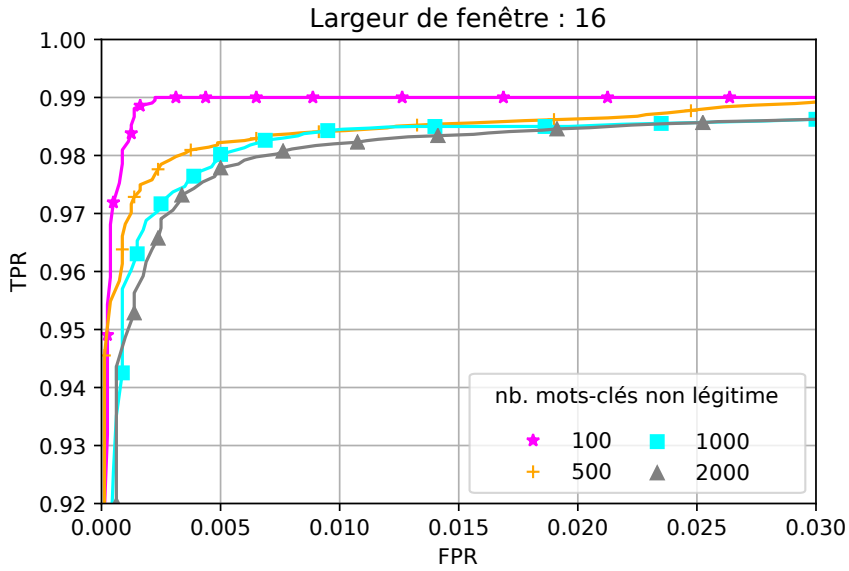


FIGURE 4.5 – Impact du nombre de mots-clés non légitimes considérés sur les résultats de la classification ($data_1$)

Impact du nombre de signatures

La dernière expérience de cette section mesure l'impact du nombre de signatures considérées (nombre de mots-clés non légitimes supervisés) par la solution sur les résultats de la classification. La classification est évaluée avec différents nombres de mots-clés non légitimes : 100, 500, 1000 ou 2000. De plus, nous utilisons 8000 mots-clés légitimes pour les tests. La largeur est fixée à $h = 16$ conformément aux conclusions de l'expérimentation précédente et α varie pour tracer une courbe ROC. Nous effectuons trois évaluations indépendantes avec sélection aléatoire des mots-clés et des traces. Les résultats de cette expérience sont tracés sur le graphique de la figure 4.5. Cette dernière souligne que l'efficacité de la classification décroît si le nombre de mots-clés non légitimes à superviser augmente. En effet, plus la liste est longue, plus la probabilité de collision augmente entraînant ainsi plus de mauvaise identification.

Finalement, les résultats obtenus suite à l'étude de l'impact des paramètres h et α sur les différentes métriques sont rassemblés dans le tableau 4.2. Il est intéressant de constater que le taux WCR est presque systématiquement nul sauf dans le cas où le seuil de filtrage est très faible. Cela signifie que, lorsqu'une trace est reconnue comme associée à un mot-clé non légitime, notre solution a une très forte probabilité d'identifier le mot-clé exact. Dans les meilleures conditions, avec $h = 16$ et $\alpha = 4,1$, on atteint un taux de justesse de 99,59% avec $TPR = 0,968$ et $FPR = 0,001$ (en gras dans le Tableau 4.2).

L'analyse des différents paramètres étant réalisée, notre objectif est d'appliquer ces résultats sur une expérimentation qui va prendre en compte l'ensemble du jeu de données que nous avons présenté dans la section 4.2 afin d'évaluer plus largement l'applicabilité de notre solution.

4.3.3 Expérience à grande échelle

Nous souhaitons maintenant observer comment se comporte la solution avec un nombre important de mots-clés non légitimes et d'observer l'impact du ratio entre les traces légitimes et

(a) TPR et FPR

h / α	1	4	16	50	1	4	16	50
	TPR				FPR			
3,5	0,832	0,983	0,989	0,948	0,0909	0,0788	0,068	0,0602
3,9	0,824	0,977	0,983	0,830	0,0182	0,0123	0,0098	0,0083
4,1	0,819	0,970	**0,968**	0,607	0,0026	0,0018	**0,001**	0,001
4,25	0,797	0,941	0,868	0,185	0,0002	0	0	0
4,35	0,464	0,223	0,007	0	0	0	0	0

(b) Justesse et WCR

h / α	1	4	16	50	1	4	16	50
	Justesse				WCR			
3,5	0,9011	0,9276	0,9379	0,9406	0,015	0,003	0	0
3,9	0,9654	0,9866	0,9895	0,975	0			
4,1	0,979	0,9953	**0,9959**	0,9584	**0**			
4,25	0,9789	0,9939	0,9863	0,9155	0			
4,35	0,9445	0,9194	0,897	0,8963	0			

Légende : 0 0,25 0,5 0,75 1 (TPR, Justesse) (FPR, WCR)

TABLE 4.2 – Résultats $data_1$: 1050 mots-clés non légitimes / 8000 légitimes

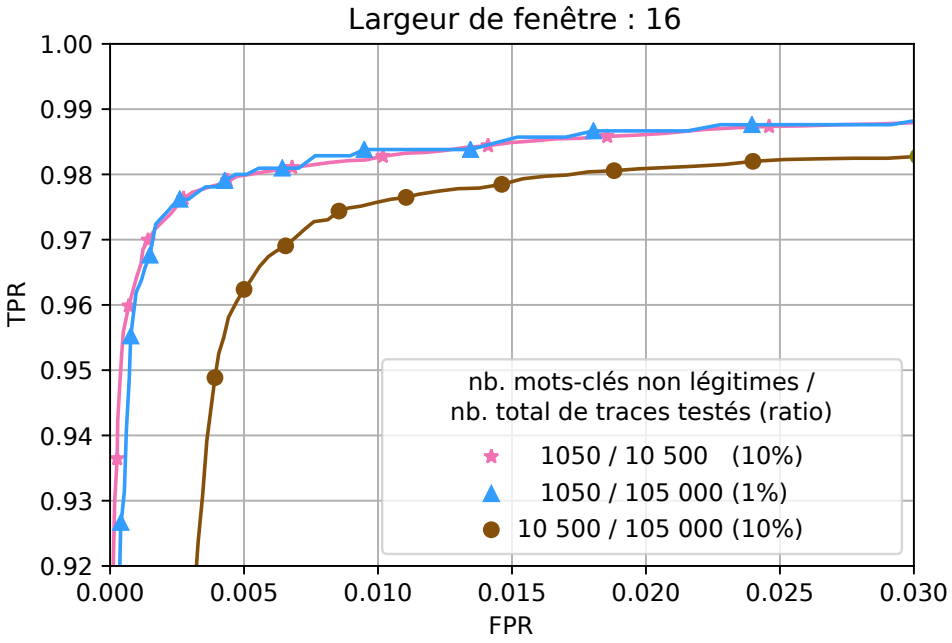


FIGURE 4.6 – Courbe ROC pour le test à grande échelle ($data_1$ et $data_2$)

(a) TPR et FPR

α \ h	1	4	16	50	1	4	16	50
	TPR				FPR			
3,5	0,820	0,979	0,986	0,927	0,1042	0,0908	0,0811	0,0739
3,9	0,814	0,973	0,980	0,790	0,0219	0,0201	0,0162	0,013
4,1	0,805	0,965	**0,966**	0,557	0,0052	0,0069	**0,0056**	0,0026
4,25	0,780	0,932	0,871	0,170	0,0009	0,0038	0,0026	0,0003
4,35	0,448	0,253	0,009	0	0	0,0004	0	0

(b) Justesse et WCR

α \ h	1	4	16	50	1	4	16	50
	Justesse				WCR			
3,5	0,8889	0,9207	0,9250	0,9262	0,017	0,001	0,001	0,001
3,9	0,9631	0,9788	0,9834	0,9691	0,004	0	0	0
4,1	0,9775	0,9884	**0,9918**	0,9574	0,001	0	**0**	0
4,25	0,9792	0,9857	0,9858	0,9242	0			
4,35	0,9498	0,8762	0,9099	0,9091	0			

Légende : 0 0,25 0,5 0,75 1 (TPR, Justesse) (FPR, WCR)

TABLE 4.3 – Résultats $data_1$ & $data_2$: 10 500 mots-clés non légitimes / 105 000 légitimes

non légitimes. Nous combinons 10 500 traces de mots-clés non légitimes (1 trace parmi les 5 disponible pour chaque mot-clé) du jeu de données $data_1$ et les traces légitimes de $data_2$ (1 trace pour 105 000 mots-clés différents). Cela nous permet ainsi de tester notre solution sur une plus grande échelle. La construction des 10 500 signatures associées aux mots-clés non légitimes s'effectue avec les quatre traces restantes pour chaque mot-clé de $data_1$.

Les résultats obtenus sont présentés sur le graphique de la figure 4.6. La performance de la classification avec le jeu de données complet apparaît en marron : ce jeu de données regroupe 10 500 mots-clés non légitimes et 105 000 mots-clés légitimes (ratio de 10%). En utilisant les paramètres définis comme optimaux dans la partie précédente, nous obtenons un score de justesse de 99,18% avec $TPR = 0,966$ et $FPR = 0,0056$ (cf. tableau 4.3). Il est intéressant de constater qu'avec un nombre constant de mots-clés non légitimes, si le ratio de traces légitimes/non légitimes varie, les performances de la classification ne change pas comme l'indiquent les courbes rose (ratio 10%) et bleu (ratio 1%). Cependant, plus le nombre de mots-clés non légitimes à superviser est important, plus les résultats se dégradent. En effet, nous pouvons remarquer sur la figure 4.6 que la courbe rose (ratio de 10%), est au-dessus de la courbe marron (ratio 10% également), présentant dix fois plus de mots-clés à superviser.

On peut conclure que le ratio entre les traces légitimes et non légitimes n'a pas d'impact sur les résultats, contrairement au nombre de mots-clés non légitimes que l'on souhaite superviser. Plus il y a de mots-clés à superviser et plus les résultats sont susceptible de baisser. Cependant au vu des résultats obtenus avec plus de 10 000 mots-clés non légitimes, notre solution est capable de supporter un nombre important de mots-clés.

4.3.4 Discussion

Application au temps réel

Notre solution permet de classer une nouvelle trace collectée en un temps moyen de 2,75 secondes. Le test a été réalisé sur 1000 traces différentes. Ce temps de calcul a été obtenu en utilisant uniquement un seul cœur cadencé à 2,2 GHz pour une taille de liste de 10 500 mots-clés non légitimes. La solution est implémentée en python et nous précisions qu'elle n'a pas été développée pour être particulièrement optimisée. La durée du temps de calcul est linéairement liée au nombre de mots-clés non légitimes à superviser.

Le temps de calcul précédent tient compte uniquement de la phase de classification (évaluation des signatures et filtrage) décrite dans le paragraphe 3.3.4. Dans notre cas d'étude qui n'est pas en temps réel, il faut au préalable charger en mémoire le fichier pcap, le traduire (parsing), et enfin extraire les informations. Toutes ces étapes ne sont pas représentatives d'une solution en temps réel car dans une telle situation, le chargement en mémoire et le parsing seront remplacés par une captation en direct du flux grâce à des bibliothèques bas niveau comme DPDK [39], XDP [40] ou [112].

Il est donc pertinent de ne prendre en compte que le temps de calcul de l'étape d'extraction des caractéristiques. Cette dernière est très rapide et dure en moyenne moins de deux dixième de secondes.

Au regard de ces valeurs, il est envisageable qu'une optimisation du code permette un usage en temps réel de la solution pour un service donné en utilisant par exemple plusieurs coeurs et en redéveloppant la solution dans un langage plus performant.

Cependant, il faut rappeler que la solution a besoin de voir l'ensemble d'une trace pour effectuer sa classification. Il n'est donc pas possible dans l'état actuel de bloquer le trafic non légitime de manière préventive. Un système de log peut être mis en place et permettant d'alimenter un moteur de configuration de sécurité, par exemple si un utilisateur fait trop de requêtes non légitimes.

Stabilité des signatures

Les miniatures que nous observons sur les pages du service Google Images évoluent dans le temps. Il n'y a, a priori, pas de règle stricte mais il y a des changements qui s'opèrent plus ou moins rapidement en fonction de la popularité du mot-clé et de l'actualité. Pour faire face à ces changements, il est possible de collecter des traces au cours du temps afin d'observer l'évolution des signatures et savoir quand la signature actuelle doit être mise à jour. Il est intéressant de noter que, pour le service Google Images en particulier, nous n'avons pas constaté de comportement dépendant de l'historique des utilisateurs. Pour autant, certaines options du service ont des impacts sur les résultats renvoyés par le serveur, comme par exemple l'option "SafeSearch". Cette dernière filtre une partie des résultats et modifie donc la signature qui en découle. Pour intégrer ces cas, il est nécessaire de considérer la requête avec l'option comme un nouveau mot-clé à apprendre. Il est aussi possible qu'un utilisateur puisse utiliser des synonymes de mots-clés non légitimes. Comme Google Images est très précis au niveau des résultats qu'il produit, la signature d'un synonyme est différente. Ainsi, pour gérer ces cas, nous pourrions recourir à l'usage de techniques du traitement automatique du langage naturel afin d'étendre la liste des mots-clés non

39. Data Plane Development Kit : `https://www.dpdk.org/`
40. eXpress Data Path : `https://www.iovisor.org/technology/xdp`

légitimes. Par exemple, l'outil DISCO [41] a montré son efficacité dans la prévention du phishing en étendant les blacklists DNS via l'analyse sémantique [113].

Limites de la méthode

Il est intéressant de se questionner sur les limites de la méthode et, en particulier, d'anticiper les cas où H1Classifier ne fonctionnerait pas. Un des éléments qui pourrait nuire de façon importante aux résultats de notre solution serait l'usage généralisé du padding comme toutes les solutions qui utilise la taille des payload. Comme nous l'avons évoqué dans le paragraphe 3.3.3, nous observons de légères variations dans les tailles que l'on récupère et KDE nous permet de les gérer. Cependant, si les variations sont trop importantes avec l'introduction massive de padding aléatoire dans les record TLS, cela impacterait négativement notre solution. En effet, le padding rendrait chaque taille d'image plus variable (avec une marge d'erreur) et entraînerait en conséquence plus de collisions.

Comme nous l'avons expliqué dans le paragraphe 3.3.4, la solution H1Classifier a besoin d'accéder à la taille des images ou, a minima, à la taille des objets HTTP. La segmentation du trafic du serveur se fait via la détection en sens inverse des requêtes du client. Cette supposition est a priori valide pour du trafic HTTP/1.1, comme nous l'avons montré auparavant. Cependant, certaines options peuvent modifier le comportement du protocole. C'est le cas du pipelining HTTP. Cette option de HTTP/1.1, présentée dans le paragraphe 1.4, a été développée pour réduire la latence du protocole. Bien que les requêtes puissent, avec cette option, être envoyées en parallèle, le serveur traitera les demandes par leur ordre d'arrivée et ne mélangera pas les paquets réponses. Ainsi, il reste possible d'obtenir nos informations en retravaillant la méthode d'extraction des tailles d'objets et de leur requête associée.

Cependant, il est important de noter que cette option n'a jamais été réellement adoptée à cause de contraintes liées aux proxys ou au blocage en tête de ligne (cf. paragraphe 1.4.2). Le pipelining a été supporté dans les navigateurs Chrome et Firefox entre 2014 [42] et 2017 [43] respectivement, mais jamais activé par défaut. Sur Internet Explorer, cette option n'a jamais été supportée.

Ainsi, l'option pipelining n'est pas un réel obstacle pour notre solution, contrairement aux techniques de multiplexage telles qu'utilisées dans le protocole HTTP/2. Ce dernier est différent du pipelining et l'assemblage des paquets pour reformer les objets HTTP ne fonctionne plus (cf paragraphe 1.4.2).

4.4 Classifier du trafic HTTP/2 avec H1Classifier

Cette section s'intéresse à l'applicabilité de la méthode de classification H1Classifier pour le trafic HTTP/2. Nous recherchons tout d'abord une nouvelle caractéristique afin d'appliquer notre solution puis nous évaluons la méthode H1Classifier avec ces nouvelles conditions. Nous discutons ensuite les résultats obtenus et analysons les raisons des mauvaises performances de H1Classifier. Finalement, nous exposons les limites de notre solution pour le trafic HTTP/2.

41. `http://www.linguatools.de/disco/disco_en.html`
42. `https://bugs.chromium.org/p/chromium/issues/detail?id=364557`
43. `https://bugzilla.mozilla.org/show_bug.cgi?id=1340655`

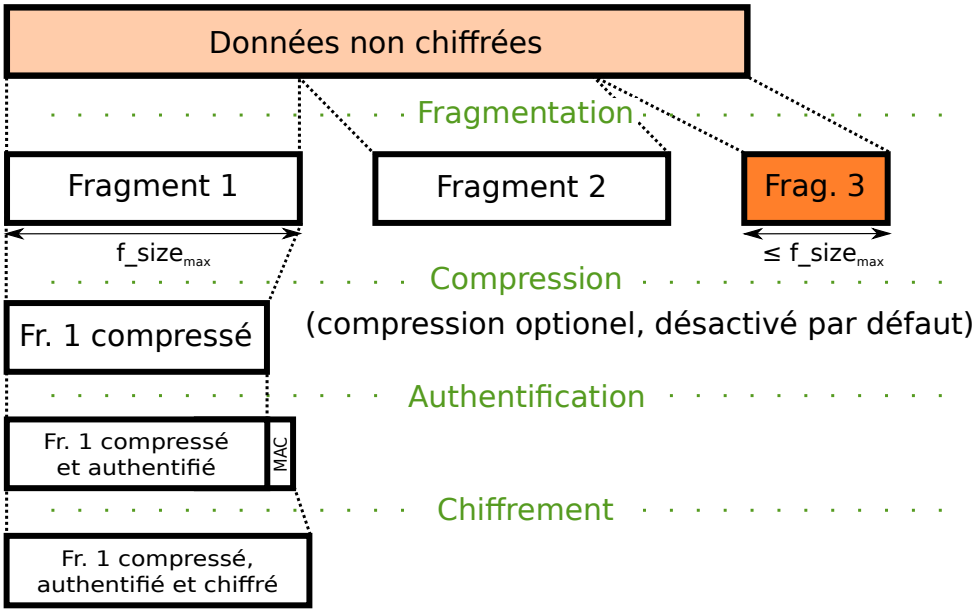


FIGURE 4.7 – Protocole record TLS

4.4.1 Recherche d'une nouvelle caractéristique

Le protocole HTTP/2 nous confronte à l'usage du multiplexage qui a pour effet de concentrer sur une seule connexion TCP l'ensemble du trafic entre le client et le serveur. De plus, plusieurs requêtes peuvent, à présent, être envoyées et traitées en parallèle. Pour rappel, une présentation étendue du protocole HTTP/2 est disponible dans la section 1.4.

Cependant, pour appliquer notre solution H1Classifier, il est nécessaire d'avoir accès à la liste des tailles d'images ou a minima à la liste des tailles des objets HTTP chiffrés. Or, cette liste n'est plus accessible avec du trafic HTTP/2 comme l'alternance entre requêtes et réponses n'est plus assurée. Aucune garantie sur l'ordre des réponses du serveur n'existe et les réponses peuvent même être entrelacées. Par conséquent, il n'est pas possible de grouper les records d'une même réponse entre eux. Il est donc nécessaire de trouver une autre caractéristique équivalente pour appliquer la méthode.

Pour cela, nous nous intéressons particulièrement à la génération des records par le protocole TLS (cf. paragraphe 1.2.2). Le schéma de la figure 4.7 résume les différentes étapes du processus. La première étape est la fragmentation : elle prend en entrée le message à transmettre en clair (de taille t_msg) et fragmente le message en fragments plus petits qui pourront être ensuite, authentifiés, chiffrés et enfin envoyés. Durant cette étape, le message est découpé en n fragments. Les $n-1$ premiers fragments ont une taille de bloc égale à une constante (f_size_{max}) définie au niveau du serveur web. Elle ne peut pas dépasser 2^{14} octets. Le dernier bloc n a, quant à lui, une taille $f_size_{max} < 2^{14}$. La taille de ce bloc n est intrinsèquement liée au message initial. Cette taille est donc très corrélée à la taille initiale du message. Par exemple, dans notre schéma, le fragment 3 qui correspond au fragment n a une taille égale à $t_msg \bmod f_size_{max}$.

Comme expliqué dans le chapitre précédent, aucune compression n'est appliquée au niveau de TLS. De plus, le chiffrement ne modifie pas la taille du message en clair, à la taille de l'empreinte d'authentification près (qui est de taille constante). On peut donc observer simplement les records

TLS et uniquement conserver ceux d'une taille inférieure à f_size_{max} pour appliquer un premier filtrage.

Cependant, un message avant fragmentation ne correspond pas nécessairement à un unique objet HTTP. Il est en théorie possible que plusieurs objets HTTP se retrouvent dans le tampon (buffer) du protocole TLS et soient placés les uns à la suite des autres et seulement ensuite fragmentés.

Ainsi, en appliquant uniquement un filtrage pour les records de taille différente de f_size_{max}, on obtient des données non représentative (justesse proche de 0). Pour pallier ce problème, nous avons introduit un nouveau paramètre de filtrage que nous nommons β. Cela permet d'éliminer toutes les tailles de records dont la fréquence d'apparition est supérieure à β. Nous effectuons ce filtrage pour conserver les tailles qui sont les moins fréquentes au sein d'une trace et qui sont donc plus susceptibles de la caractériser.

Après avoir reconstruit la séquence des tailles de records TLS, nous appliquons un filtrage sur ces tailles en fonction de leur taux d'apparition. Ce filtrage est défini comme le produit de la fréquence d'apparition moyenne de l'ensemble des tailles (calculée indépendamment pour chaque service) avec le facteur β. Le paramètre β est une constante à déterminer pour chaque service lors de l'évaluation. La définition du seuil de filtrage s'écrit :

$$seuil_filtrage = \beta \times moyenne(liste_frequences_tailles) \tag{4.1}$$

Ce filtrage permet de mettre de côté les tailles trop fréquentes représentatives des objets HTTP communs entre les différentes pages ou à des paquets de contrôle. Cette étape écarte par la même occasion les records de taille égale à f_size_{max}.

4.4.2 Évaluation de H1Classifier avec du trafic HTTP/2

Pour évaluer les signatures produites par KDE pour HTTP/2, on considère une classification en monde fermé. Cela signifie que nous allons uniquement tester des mots-clés connus durant l'entraînement. C'est un scénario optimiste par rapport au scénario en monde ouvert que nous avons utilisé dans la section 4.3, mais qui permet une évaluation rapide de l'intérêt ou non de la méthode.

Nous utilisons un jeu de données composé de 2000 mots-clés avec 5 traces pour chaque mot-clé pour 5 services différents (Amazon, Instagram, Google, Google Images et Google Maps). Ce jeu de données est décrit plus loin dans la section 6.2.1. Les conditions d'application de la solution H1Classifier sont similaires à celles de la section 4.3, à l'exception de la caractéristique utilisée. Dans cette expérience, nous n'utilisons plus la taille des images comme caractéristique pour l'apprentissage des signatures, mais nous utilisons directement la taille des records TLS si la taille a une fréquence d'apparition inférieure au seuil de filtrage défini dans l'équation (4.1).

Pour l'évaluation, nous avons au préalable fixé les valeurs des paramètres de H1Classifier, à savoir h la fenêtre pour KDE et β le seuil de filtrage de records par la fréquence. Le paramètre α n'intervient pas ici. En effet, dans le cas d'un scénario en monde fermé, une signature existe pour tout mot-clé (pas de mots-clés légitimes) or α permet de définir si nous sommes en présence d'un mot-clé non légitime.

TABLE 4.4 – Résultats pour H1Classifier avec du traffic HTTP/2 en monde fermé (moyenne 5 évaluations)

Source	Amazon	Instagram	Google	Google Images	Google Maps
β, h	1 ; 0,5	0,5 ; 0,5	0,1 ; 0,5	1 ; 0,5	1 ; 0,5
Justesse	0,8475	0,5865	0,259	0,9914	0,924
écart-type	0,0076	0,0052	0,0071	0,0013	0,0137

Le Tableau 4.4 résume l'ensemble des résultats obtenus pour les meilleures valeurs des paramètres. Ces valeurs ont été obtenus expérimentalement au préalable. Chaque résultat est donné sous la forme de la justesse moyennée sur cinq évaluations indépendantes avec l'écart type correspondant.

La valeur optimale obtenu expérimentalement pour h est de $0,5$. Cela signifie que l'utilité de KDE est ici très limitée. En effet, un paramètre de lissage de valeur $h = 0,5$ signifie que le lissage s'opère sur une très petite fenêtre, inférieure à une unité. Or, les tailles que nous traitons sont entières (en octets). Cela signifie que les tailles que nous utilisons avec HTTP/2 ne varient pas comme celles utilisées pour HTTP/1. L'usage de KDE devient donc obsolète car son intérêt est d'endiguer ces variations dans les tailles.

Concernant le paramètre β, on observe qu'il est dépendant du service considéré et qu'il est donc nécessaire de l'apprendre pour chacun des nouveaux services étudiés.

Au niveau des résultats pour la justesse, on voit des résultats satisfaisants (plus de 90%) pour les services Google Images et Google Maps. Cependant, pour les autres services, les résultats sont moins encourageants et font apparaître la nécessité d'avoir une solution plus générique pour le support de plusieurs services.

4.4.3 Limitations du portage de la méthode

Nous décrivons ici les limites de la solution H1Classifier, lorsqu'elle est appliquée à du trafic HTTP/2, dont découle l'efficacité amoindrie décrite précédemment.

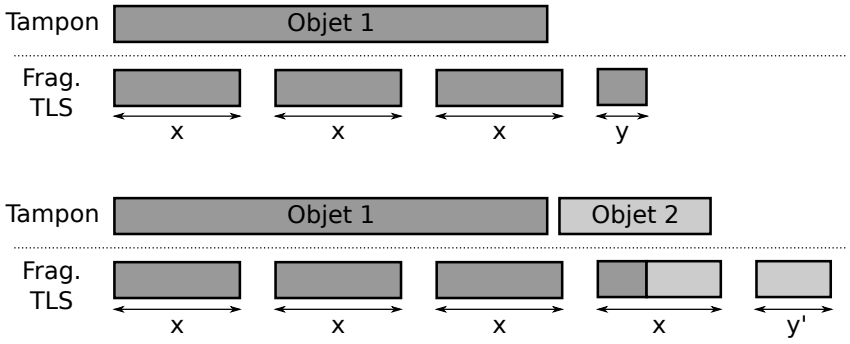


FIGURE 4.8 – Exemple de fragmentation avec un ou deux objets dans le tampon d'entrée du protocole TLS

Premièrement, avec HTTP/2, il existe plusieurs types de frames et les objets HTTP de type *data frame* (ceux qui contiennent des données utiles) peuvent cohabiter entre eux ou même avec d'autres types dans le tampon. Lorsque la fragmentation survient, les différents objets HTTP

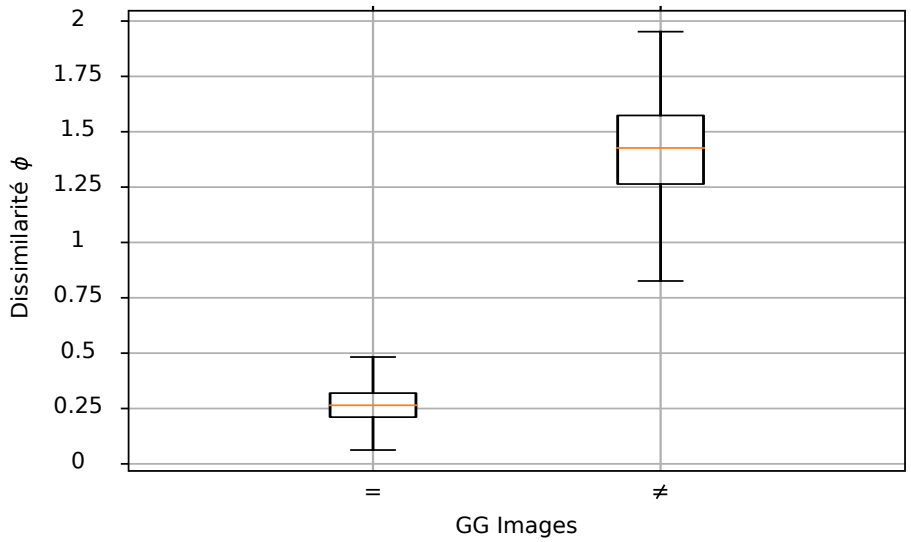


FIGURE 4.9 – Calcul de dissimilarité pour des fonctions signatures avec du trafic HTTP/1.1(1000 mots-clés) sur le service Google Images

sont découpés les uns à la suite des autres et un record peut contenir un morceau de plusieurs objets. Dans la figure 4.8, on peut voir un exemple lorsque le tampon d'entrée de TLS contient un ou deux objets différents. Dans le premier cas, le dernier fragment est représentatif de l'objet initial. Mais, dans le deuxième cas, le dernier fragment est, quant à lui, représentatif de l'agrégation des deux objets en même temps. Lorsqu'il y a agrégation, la taille caractéristique que l'on récupère donne moins d'information qu'une taille d'objet. De plus, il est important de noter qu'il est possible de rencontrer les deux cas décrits précédemment lors de deux chargements distincts de la même page. Ainsi, augmenter le nombre de traces d'apprentissage peut aider à contrebalancer cet effet car la signature sera alors plus générale grâce aux différents exemples déjà rencontrés et appris.

Deuxièmement, en pratique, il est fréquent que la taille f_size_{max} soit inférieure à la taille du MTU (autour de 1500 octets). Cette valeur est fixée par le serveur web [114] et peut même varier et évoluer en fonction de l'état de la connexion, comme le fait TCP avec le *slow start* pour la taille de la fenêtre. La taille maximale des records a deux impacts antagonistes sur la classification dont il est difficile d'en prédire l'impact :

— D'un côté, plus le f_size_{max} est élevé, plus le nombre de tailles possibles est grand et donc plus le nombre de collisions possibles est réduit.

— D'un autre côté, plus le f_size_{max} est élevé, plus il est probable d'avoir plusieurs objets concaténés dans le tampon d'entrée de TLS.

4.4.4 Test du caractère discriminant des signatures

Afin de comprendre les résultats produits précédemment, nous évaluons ici le caractère discriminant des signatures. Nous cherchons à quantifier numériquement la représentativité de la signature associée à un mot-clé. Pour cela, nous allons calculer un score de dissimilarité pour chaque paire de signatures. Comme les signatures sont des fonctions, le score de dissimilarité entre deux signatures correspond à l'aire située entre les deux courbes représentatives de ces

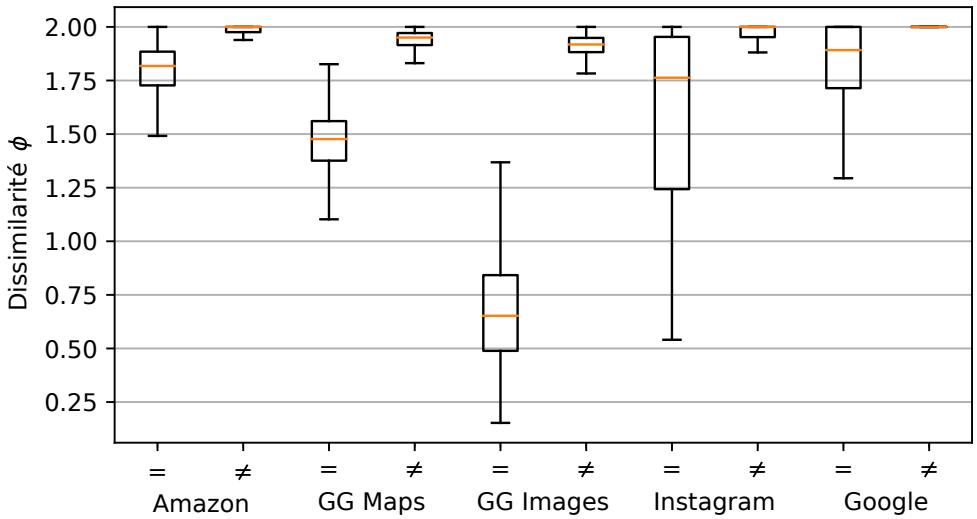


FIGURE 4.10 – Calcul de dissimilarité pour des fonctions signatures avec du trafic HTTP/2

fonctions. Formellement, nous définissons ϕ le score de dissimilarité comme suit :

$$\phi(m_1, m_2) = \int_0^\infty \mid \sigma_{m_1}(x) - \sigma_{m_2}(x) \mid \mathrm{d}x \tag{4.2}$$

avec σ_{m_1} et σ_{m_2} les fonctions signature des mots-clés respectifs m_1 et m_2.

Le score de dissimilarité ϕ est compris entre 0 et 2 car $\forall m : \int_0^\infty \sigma_m(x)\,\mathrm{d}x \leq 1$ comme σ_m est une fonction de densité. Nous rappelons que la fonction de densité est décrite dans la section 3.3.3. Plus le score de dissimilarité est élevé, plus les signatures sont différentes.

A cause de la combinatoire élevée des tests deux à deux qui croît très rapidement, nous avons limité le nombre de mots-clés considérés à 1000. Ces derniers sont sélectionnés de façon aléatoire. Pour l'ensemble des paires parmi ces 1000 mots-clés, nous calculons le score de dissimilarité donné par l'équation (4.2). Nous comparons le score de dissimilarité calculé pour deux signatures de deux mots-clés identiques (même mot-clé mais traces différentes), et aussi deux mots-clés différents.

Dans un premier temps, nous présentons les résultats obtenus pour du trafic HTTP/1.1 sur le service Google Images sur le diagramme de la figure 4.9. Nous rappelons que nous étudions uniquement ce service avec HTTP/1.1 car c'est le pire cas (cf. paragraphe 3.2.2). Comme nous pouvons le voir sur la figure, les signatures sont hautement discriminantes lorsque les mots-clés étudiés sont différents. On ne remarque pas de recouvrement pour les diagrammes en boîte car les signatures permettent aisément de distinguer deux mots-clés différents et, si les mots-clés sont identiques, alors la dissimilarité est relativement faible, même pour les cas les plus extrêmes. Il est intéressant de noter que l'écart type du score de dissimilarité dans le cas des mots-clés identiques est important. Cela s'explique par le fait que les tailles des objets que nous utilisons pour les signatures sont variables comme nous l'avons constaté dans le paragraphe 3.3.3. L'usage de plusieurs traces nous permet d'apprendre les variations dans les signatures. Ce qui confirme une fois de plus l'intérêt de notre approche basée sur KDE.

Pour le trafic HTTP/2, les résultats sont reportés sur la figure 4.10. Les résultats pour les services Google Images et Google Maps donnent des diagrammes en boîte, bien disjoints entre les mots-clef identiques et les mots-clef différents. Cela est en accord avec les résultats obtenus dans le tableau 4.4 car il s'agit des deux services qui présentaient les meilleurs scores. Pour les autres services, les diagrammes en boîte se superposent et donc la valeur de score n'est pas discriminante à 100% d'où les résultats moins bons pour ces derniers services dans le tableau 4.4.

4.5 Synthèse

H1Classifier est l'implémentation de notre méthode de classification décrite dans le chapitre 3. Dans ce chapitre nous avions deux objectifs : évaluer H1Classifier sur du trafic HTTP/1.1 puis étudier son adaptation à du trafic HTTP/2.

L'évaluation de H1Classifier a été effectuée selon un scénario en monde ouvert sur le service Google Images. Dans un premier temps nous avons construit le jeu de données permettant cette évaluation puis dans un deuxième temps nous avons réglé les paramètres de la classification (h la largeur de la fenêtre pour KDE et α le seuil de filtrage) avec un sous ensemble du jeu de donnée.

Finalement nous avons évalué H1Classifier dans le cadre d'un scénario contenant 10 500 mots-clés non légitimes ainsi que 105 000 mots-clés légitimes. Lors de cette expérience qui teste le passage à l'échelle de la méthode nous avons obtenu un taux de justesse de 99,18% (TPR = 0,966 et FPR = 0,0056). Ces résultats montrent ainsi, la très bonne performance de notre solution H1Classifier pour détecter l'usage de mots-clés non légitimes et leur reconnaissance, parmi de nombreuses traces de mots-clés légitimes.

Concernant l'adaptation de H1Classifier au trafic HTTP/2, la conclusion est que cette méthode n'est pas adaptée à ce type de trafic pour les raisons suivantes. Premièrement, nous utilisons KDE avec une fenêtre de $h = 0,5$. Cela signifie que les tailles de records TLS ne varient pas autant (entre deux requêtes pour un même mot-clé) avec du trafic HTTP/2 qu'avec du trafic HTTP/1.1. Comme KDE était principalement utilisé pour réduire les variations observées sur le trafic HTTP/1.1, son usage n'a *a priori* plus lieu d'être. Deuxièmement, nous avons dû introduire le paramètre β pour HTTP/2 qui dépend du service considéré et rend la solution moins simple à mettre en place. Troisièmement, comme le trafic HTTP/2 ne nous permet pas de distinguer ni le lien entre les records d'un même objet ni la requête à laquelle ils sont associés, nous ne pouvons pas obtenir la taille des objets HTTP. Nous avons trouvé une caractéristique qui peut être une alternative mais est moins précise car le spectre des valeurs possibles est réduit. Comme nous ne pouvons pas filtrer les tailles pour ne garder que celles associées à des images, nous avons plus de valeurs et donc plus de bruit. Cela réduit l'efficacité de la classification et allonge les temps de calcul, comme on peut le comprendre avec l'équation (3.10) qui donne le fonctionnement de KDE. Finalement, si nous filtrons les tailles qui apparaissent fréquemment nous perdons une partie des informations. Pour autant, si nous les laissons toutes, cela dégrade grandement les résultats de la classification à cause du bruit engendré.

Pour ces raisons, nous considérons que l'usage de la méthode H1Classifier pour du trafic HTTP/2 n'est pas une solution satisfaisante et nous présentons dans le chapitre suivant les travaux que nous avons menés pour trouver une classification plus efficace dans ce cas.

Troisième partie

Analyse de trafic chiffré HTTP/2

Chapitre 5

Modélisation et méthode pour l'analyse du trafic chiffré HTTP/2

Sommaire

5.1 Introduction 84
5.2 Modélisation du trafic chiffré HTTP/2 84
5.2.1 Modélisation initiale 84
5.2.2 Présentation des caractéristiques 85
5.2.3 Modélisation raffinée 87
5.3 Méthode de classification pour le trafic chiffré HTTP/2 87
5.3.1 Vue d'ensemble 88
5.3.2 Extraction des caractéristiques à partir d'un flux réseau 89
5.3.3 Classification par apprentissage supervisé 90
5.3.4 Forêt d'arbres décisionnels 91
5.4 Synthèse 94

5.1 Introduction

Comme vu à la fin du chapitre 4, la modélisation du trafic HTTP/2 avec la taille des objets HTTP n'est plus possible. Il nous faut donc définir un nouveau modèle de trafic plus spécifique au protocole HTTP/2 utilisant un ensemble de caractéristiques du trafic qui permet la détection malgré le chiffrement, et ensuite construire une méthode de classification reposant sur ces caractéristiques. Nous avons utilisé une solution d'apprentissage supervisé car les caractéristiques que nous collectons sont plus variables que pour HTTP/1.1. En effet, il n'est plus possible de réaliser de la reconnaissance de motifs suite à l'agrégation des informations issues des caractéristiques de plusieurs traces.

Dans une première section, nous présentons une modélisation haut niveau du trafic HTTP/2 en revenant à la notion de records TLS à partir desquels nous définissons une liste de caractéristiques du trafic. Nous raffinons ensuite le modèle en prenant en compte ces caractéristiques.

Dans une deuxième section, nous détaillons notre méthode de classification du trafic. Après une vue d'ensemble qui met en avant les phases de génération du modèle et de supervision, nous en décrivons les différentes étapes. Afin de réaliser une classification considérant l'ensemble des caractéristiques définies, nous introduisons les techniques d'apprentissage automatique et en particulier l'apprentissage supervisé. Nous présentons différents algorithmes existants et motivons notre choix pour les forêts d'arbres décisionnels dans le cadre de notre approche.

5.2 Modélisation du trafic chiffré HTTP/2

Nous donnons ici un aperçu des informations disponibles lors de notre analyse des traces réseau associées à la requête d'un mot-clé sur un service qui utilise le protocole HTTP/2 chiffré. Dans la suite, HTTP/2 fera référence implicitement à son usage couplé à TLS.

5.2.1 Modélisation initiale

Une modélisation haut niveau du trafic HTTP/2 est semblable à celle du trafic HTTP/1.1. En effet, le protocole HTTP/2 étant toujours transporté par TCP, on peut donc encore avoir accès à la taille de la charge utile (payload) de chaque paquet. Le trafic est également chiffré grâce au protocole TLS, il est donc possible pour chaque trace collectée de produire un ensemble de records associé au mot-clé qui l'a générée. Ainsi, on peut reprendre la modélisation proposée pour HTTP/1.1 avec l'équation (3.2) rappelé ci-dessous :

$$\forall a \in A, \exists m \in M : a = \langle m, \{r_1, \ldots, r_i, \ldots, r_n\} \rangle \tag{5.1}$$

avec r_i les records TLS composant la trace a, associé au mot-clé m.

L'utilisation brute des records TLS et en particulier des tailles de ces derniers ne semble pas être optimale au vu des résultats obtenus dans la section 4.4. De plus, afin de raffiner le modèle des traces réseau, nous ne pouvons pas compter sur l'extraction d'information haut niveau comme ce fût le cas pour HTTP/1.1 (objets HTTP ou images). Pour le trafic HTTP/2 nous devons trouver des caractéristiques qui permettent de différencier chaque trace et construire les signatures de chaque mot-clé.



FIGURE 5.1 – Caractéristiques pour l'identification de traces HTTP/2

5.2.2 Présentation des caractéristiques

Tout d'abord, nous avons choisi de ne pas exploiter de caractéristiques temporelles, à savoir les statistiques sur les délais inter-paquets, le délai entre le premier et le dernier paquet d'une trace ou le débit (octet / temps). Nous souhaitons en effet minimiser l'impact de l'environnement sur l'identification des traces. Avec différentes positions géographiques ou différents accès réseaux, la latence sera différente et modifiera les valeurs de ces caractéristiques et donc du modèle final. Pour minimiser ces variations, les caractéristiques retenues, illustrées dans la figure 5.1, sont décorrélées du temps.

Nous avons distingué les caractéristiques usuelles, qui proviennent de l'état de l'art, des caractéristiques spécifiques définies pour nos expériences et mieux adaptées pour HTTP/2.

Caractéristiques usuelles

Les caractéristiques sélectionnées ci-dessous sont issues d'autres publications sur l'analyse de trafic chiffré, principalement des travaux de Panchenko & al. [67] et Hayes & al. [74] qui traitent

du problème de détection sur du trafic anonymisé (par exemple TOR). Elles ne sont pas spécifiques à du trafic HTTP/2 et peuvent être regroupées en trois thèmes :

(a) Statistiques de connexion (3 caractéristiques) : un décompte du nombre de records TLS (respectivement entrant ou sortant) de la trace en cours d'analyse. Une autre caractéristique quantifie les données échangées (en octet) via la somme des tailles des records TLS.

(b) Statistiques sur les "rafales" (burst) (10 caractéristiques) : deux méthodes ont été définies afin de mesurer les "rafales" ou concentration de paquets entrant (du serveur vers le client). Premièrement, nous collectons la quantité (en octets) de données envoyées par le serveur entre deux paquets émanant du client. A partir de ces données collectées nous formons une séquence de tailles et nous en extrayons : le minimum, le maximum, l'écart type, la moyenne et la médiane. La seconde méthode pour évaluer les "rafales" consiste à compter le nombre de records TLS en provenance du serveur sur chaque fenêtre de 20 records. Ensuite, nous extrayons à nouveau les mêmes statistiques que pour la méthode précédente et nous les utilisons comme caractéristiques.

(c) Nombre de tailles différentes (2 caractéristiques) : nous comptons ici simplement le nombre de tailles différentes pour les records TLS venant du serveur (records entrants) et pour ceux venant du client (records sortants).

Nouvelles caractéristiques pour HTTP/2

Pour s'adapter plus particulièrement au trafic HTTP/2, nous avons introduit de nouvelles caractéristiques suite aux expérimentations que nous avons menées dans la section 4.4 du chapitre précédent. Elles se divisent en deux groupes :

(d) Top 20 des tailles de records TLS les moins fréquentes (2×20) : en reprenant l'idée développée dans le paragraphe 4.4.1, notre but est de récupérer les tailles des records TLS représentatifs de la trace en cours. Pour cela nous extrayons l'ensemble des tailles de records TLS possibles avec leurs fréquences associées. Nous rappelons qu'une taille de record moins fréquente est probablement liée à la taille d'un objet HTTP chiffré (cf. paragraphe 4.4.1). Avec ces données nous pouvons définir une liste de 20 tailles de records TLS sortant les moins fréquentes et nous faisons le même travail pour les records TLS entrant. Cela constitue finalement un vecteur de 40 valeurs de tailles (exprimé en octet).

(e) Distribution des tailles (jusqu'à $2 \times 18\ 432$) : cette caractéristique calcule la fréquence des tailles de records TLS possibles pour les records. Afin de produire un vecteur de taille fixe nous considérons toutes les tailles potentielles de records TLS. Comme expliqué dans la section 1.2.2 sur la présentation de TLS, la taille maximale pour un record TLS est de $2^{14} + 2048$ octets. 2^{14} correspond à la taille maximale des données après fragmentation. Une marge de 1024 octets est considérée pour l'étape de compression et enfin encore 1024 octets pour les étapes d'authentification et de chiffrement cumulées.

Finalement on obtient $2 \times 18\ 432 = 36\ 864$ tailles possibles. Cela introduit théoriquement un vecteur à très forte dimension. Cependant, en pratique la plupart des valeurs ne sont pas observées lors de nos captures. Par exemple pour nos différents services considérés nous avons utilisé (pour les deux directions) 28 699 tailles pour *Amazon*, 5073 pour *Instagram*, 2165 pour *Google*, 14 769 pour *Google Images* et 17 710 pour *Google Maps*.

5.2.3 Modélisation raffinée

Nous présentons ici le raffinement du modèle d'une trace réseau HTTP/2. Afin de pouvoir produire les caractéristiques énoncées dans le paragraphe 5.2.2, il est nécessaire d'extraire pour chaque record collecté, trois éléments : la taille du record, le type de record et la direction. La taille du record s'obtient via la lecture du champs *length* (deux octets) des entêtes en clair du protocole TLS. Le type de record est obtenu de façon semblable dans le champ *content type* (1 octet). Il est intéressant de connaître le type pour garder uniquement les records de type *Application Data* qui sont liés à du contenu envoyé. Enfin nous avons la direction qui se déduit des adresses IP ou des ports source/destination. Nous pouvons à présent réécrire l'équation (5.1) de la façon suivante :

$$\forall a \in A, \exists m \in M : a = \langle m, \{(r_{l_1}, r_{t_1}, d_1), \ldots, (r_{l_i}, r_{t_i}, d_i), \ldots, (r_{l_n}, r_{t_n}, d_n)\}\rangle \tag{5.2}$$

Chaque record peut s'exprimer sous la forme d'un triplet. On note r_i un record, r_{l_i} sa taille, r_{t_i} son type et d_i sa direction. Pour rappel, A représente l'ensemble des traces réseau et M l'ensemble des mots-clés.

A partir de cette séquence de triplets, nous pouvons calculer les caractéristiques de la figure 5.1. Nous introduisons ici C l'ensemble des vecteurs de caractéristiques tel que $\forall c \in C$, $c = \{c^1, \ldots, c^j, \ldots, c^p\}$ (c étant directement dérivé de $\{(r_{l_1}, r_{t_1}, d_1), \ldots, (r_{l_i}, r_{t_i}, d_i), \ldots, (r_{l_n}, r_{t_n}, d_n)\}$), c^j est la j-ème caractéristique du vecteur c et $|C| = |A|$. On obtient ainsi l'équation ci-dessous pour une trace HTTP/2 :

$$\forall a \in A, \exists m \in M : a = \langle m, \{c^1, \ldots, c^j, \ldots, c^p\}\rangle \tag{5.3}$$

avec p le nombre de caractéristiques.

Notre objectif est de trouver une fonction f qui est capable de distinguer, à partir d'un ensemble de caractéristiques issues d'une trace, si la trace est associée à un mot-clé légitime ou non. Pour cela, nous pouvons décrire le problème comme suit :

$$\begin{aligned} \forall c \in C,\ c &= \{c^1, \ldots, c^j, \ldots, c^p\}, \\ f(c) &= \begin{cases} m \in M_{nl} \\ leg \end{cases} \end{aligned} \tag{5.4}$$

avec c un vecteur de caractéristiques lié à un mot-clé m, M_{nl} l'ensemble des mots-clés non légitimes et leg la réponse de la fonction f dans le cas d'un mot-clé légitime.

5.3 Méthode de classification pour le trafic chiffré HTTP/2

Nous rappelons que nous souhaitons pouvoir prédire le mot-clé sous-jacent à une trace réseau si cette dernière correspond à un mot-clé non légitime. Pour cela nous devons apprendre à reconnaître ces traces non légitimes en extrayant des caractéristiques de ces dernières et en construisant un modèle capable de reconnaître les caractéristiques identifiables. Nous avons eu recours aux méthodes d'apprentissage automatique et en particulier à l'algorithme de forêt d'arbres décisionnels afin d'établir notre classification.

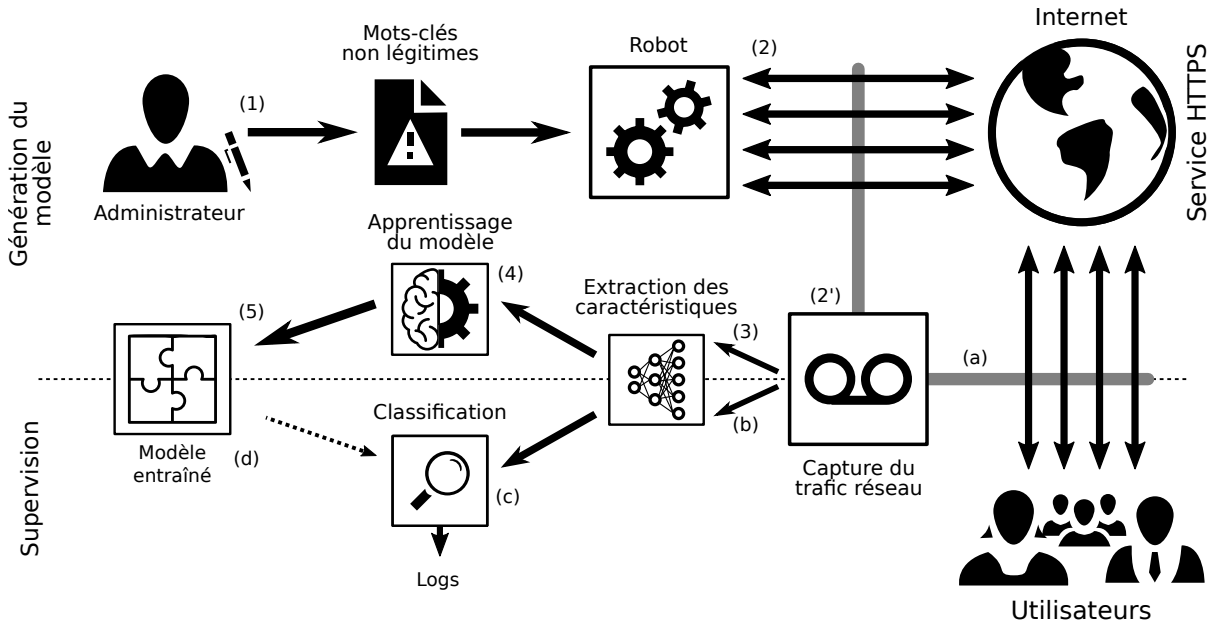


FIGURE 5.2 – Architecture globale de la solution (trafic HTTP/2)

Nous commençons cette section par une vue d'ensemble de la solution de classification du trafic. Ensuite, nous détaillons l'étape d'extraction des caractéristiques, puis nous discutons les solutions d'apprentissage automatique pour enfin introduire l'algorithme des forêt d'arbres décisionnels (RF). Comme nous le verrons, cet algorithme nous permet de définir les étapes d'apprentissage du modèle ainsi que la partie classification.

5.3.1 Vue d'ensemble

La classification du trafic chiffré HTTP/2 se structure de façon similaire à celle présentée pour HTTP/1.1 dans le paragraphe 3.3.1. En effet, on retrouve une phase qui apprend un modèle permettant l'identification de mots-clés non légitimes puis une deuxième phase qui supervise le trafic. Pendant cette deuxième phase, nous collectons des traces inconnues et les classons afin de déterminer si elles sont associées à un mot-clé légitime ou non. L'architecture globale de la solution est schématisée dans la figure 5.2 et détaillée ci-dessous.

Pour la phase concernant la génération du modèle, l'étape (1) de définition des mots-clés non légitimes par l'administrateur est également dépendante du service considéré. L'administrateur peut définir une liste de mots-clés non légitimes pour chaque service supporté. Les étapes (2) et (2'), correspondant respectivement à l'automatisation des requêtes et à la capture des traces générées, sont identiques à celles mise en places pour le trafic HTTP/1.1 (cf. paragraphe 3.3.1). Chacune des traces est ensuite traitée (3) pour en extraire les caractéristiques nécessaires à leur identification. Cette étape est détaillée dans le paragraphe 5.3.2. Suite à l'extraction, (4) l'algorithme RF génère (5) un modèle qui pourra être interrogé afin de classifier des traces inconnues.

La phase de supervision débute toujours par (a) la collecte des traces produites par les utilisateurs supervisés. Pour chaque trace ainsi collectée, nous effectuons (b) une extraction des caractéristiques d'identification, cette étape est identique à celle de l'étape (3) de la phase de génération du modèle. Finalement, (c) la classification va interroger (d) le modèle entraîné avec les

caractéristiques de la trace en cours d'analyse. La solution classera alors la trace dans la catégorie légitime ou non légitime, déduisant le cas échéant le mot-clé (non légitime) correspondant.

5.3.2 Extraction des caractéristiques à partir d'un flux réseau

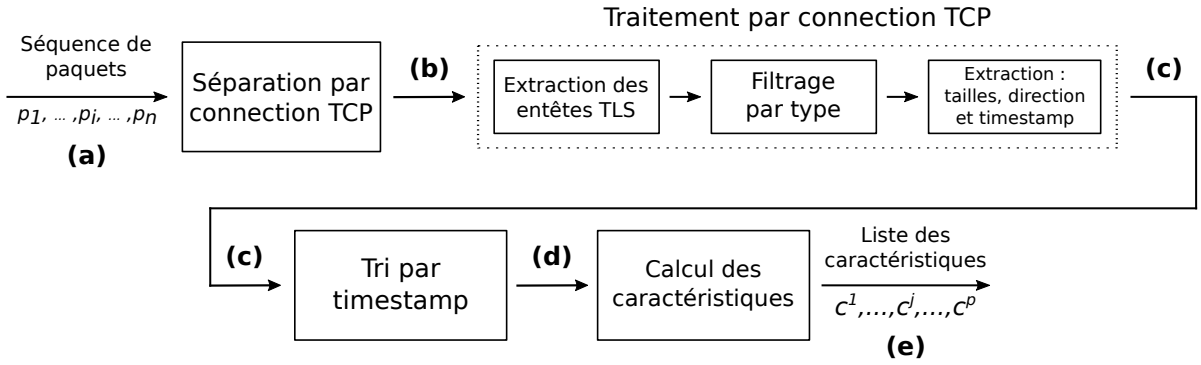


FIGURE 5.3 – Schématisation du processus d'extraction des caractéristiques pour le trafic HTTP/2

Comme illustré dans la figure 5.3, nous récupérons pour chaque trace, (a) une séquence de paquets. La séquence est ensuite séparée en fonction des différentes connexions TCP établies. Pour rappel, une connexion TCP est définie par une paire d'adresses IP (source et destination) et une paire de ports TCP (port source client aléatoire et port 443 pour les service HTTPS). Nous obtenons ainsi (b) plusieurs séquences de paquets (une par connexion TCP).

Pour chaque séquence, nous opérons plusieurs actions indépendamment. Tout d'abord nous extrayons des paquets les records TLS et en particulier leur entête (le reste étant chiffré). Dans les entêtes nous récupérons en premier le champs *type* et nous filtrons les records pour ne garder que ceux avec *type* = APPLICATION_DATA. Cela permet de ne conserver que les records transportant des données. Avec les records de données nous extrayons la taille de ces derniers avec le champs TLS *length*, la direction via l'adresse IP ou le port (en fonction de la connaissance au niveau de la sonde) ainsi que le l'horodatage (timestamp) du paquet contenant le début de l'entête du record TLS.

Nous obtenons (c) différentes séquences de plusieurs triplets (taille, direction, timestamp). Ces séquences sont ensuite fusionnées en une seule séquence grâce au tri des triplets en fonction de leur timestamp. En conséquent, nous obtenons (d) une séquence de paires (taille, direction) ordonnée chronologiquement du point de vue de la sonde (point de capture). Finalement, avec cette séquence nous calculons les valeurs des caractéristiques définies dans le paragraphe 5.2.2.

En sortie de cette étape (e) nous obtenons la liste des caractéristiques associées à une trace soit pour la phase de génération du modèle, soit pendant la phase de supervision pour les étapes (4) ou (c) illustrées dans la figure 5.2.

5.3.3 Classification par apprentissage supervisé

Pour exploiter les caractéristiques présentées précédemment, nous introduisons une technique capable de classer les différentes traces dans la catégorie légitime ou non légitime à partir d'un vecteur de caractéristiques. Pour cela nous avons utilisé des algorithmes d'apprentissage automatique et en particulier l'apprentissage supervisé.

Après avoir défini l'apprentissage supervisé et expliqué pourquoi il répond à nos besoins, nous présentons différentes méthodes existantes dans ce domaine et nous les comparons. Enfin nous détaillons le fonctionnement de l'algorithme des arbres décisionnels qui est l'algorithme que nous avons sélectionné.

Définition

L'apprentissage supervisé correspond à l'apprentissage automatique d'une fonction prédictive reposant sur des exemples préalablement étiquetés. Il y a deux catégories de problèmes dans l'apprentissage supervisé.

La première catégorie vise à prédire les valeurs d'une variable en apprenant un comportement sur la base d'exemples. Cela permet de faire des prédictions sur un déroulement futur ou d'extrapoler des valeurs à partir d'autres. Ces problèmes entrent dans la catégorie des régressions.

La deuxième catégorie de problèmes dite de classement, a pour objectif d'attribuer une classe/un label pour chaque échantillon que l'on souhaite trier. Les labels sont connus et leur profil est appris sur la base d'exemples annotés.

Notre problème défini dans le paragraphe 5.2.3 correspond exactement à la deuxième catégorie d'apprentissage supervisé. En effet, notre but est bien d'apprendre à identifier les mots-clés non légitimes grâce à des exemples afin de reconnaître à nouveau ces mots-clés non légitimes pour des traces inconnues.

Principales méthodes de classification

Les principales méthodes de classification par apprentissage supervisé peuvent être classées en cinq catégories :

— Machine à vecteurs de support (SVM : Support Vector Machine).

— Classification Naïve Bayésienne (NB)

— Arbres décisionnels (DT : Decision Trees)

— Méthode des k plus proches voisins (KNN : K-Nearest Neighbors algorithm)

— Réseaux de Neurones (NN : Neural Network)

Il est important de noter qu'il n'y a pas un meilleur algorithme a priori mais que chacun a ses propres avantages en fonction du problème rencontré. Dans notre cas les caractéristiques que nous utilisons ne sont pas pas normalisées et sont hétérogènes (compteurs, valeurs discrètes ou continue). A cause de cela, les algorithmes axés sur une notion de distance (KNN, SVM) n'ont pas été retenus. D'autre part, les caractéristiques utilisées ne sont pas indépendantes entre elles et dans ce cas les algorithmes tels que NN et DT sont plus adaptés.

Comme nous l'avons vu pour la collecte des données pour le trafic HTTP/1.1 dans la section 4.2 ou comme nous le verrons pour HTTP/2 dans la section 6.2.1, il est coûteux en temps et

en ressources de capturer des traces afin de réaliser l'apprentissage. Ainsi les méthodes nécessitant de grande quantité de données pour la phase d'apprentissage comme les réseaux de neurones ne seront pas privilégiés à cause de la difficulté pour les entraîner.

Au regard des constatations précédentes notre choix d'algorithme s'est orienté vers les DT. Les arbres de décisions ont en outre le bénéfice d'être peu paramétrés et ne nécessitent donc pas d'optimisation complexe. De plus, dans le cas de l'analyse de trafic réseau chiffré, des travaux comme [44, 74] font usage également des arbres décisionnels sous la forme de forêts. La méthode des forêts d'arbres décisionnels (RF pour random forest) utilise plusieurs arbres de décisions et permet souvent de généraliser plus facilement le modèle [115]. Pour l'ensemble de ces raisons, nous utilisons cet algorithme pour apprendre à identifier les traces correspondant à nos mots-clés non légitimes.

5.3.4 Forêt d'arbres décisionnels

Les forêts d'arbres décisionnels (RF) [116] sont la continuité des travaux de Breiman *et al.* sur les arbres de décision et en particulier la méthode CART (ClAssification and Regression Trees) [117]. En effet RF est une extension des algorithmes DT. Le principe est d'utiliser plusieurs DT qui sont chacun entraînés sur une partie des données. Ensuite pour classer un nouvel échantillon, chaque arbre est évalué indépendamment avec les caractéristiques de l'échantillon et un vote par majorité est opéré pour désigner l'étiquette à adopter.

Dans ce paragraphe nous présenterons la version de l'algorithme RF utilisé par la librairie scikit-learn [111] qui diffère légèrement de la version de Breiman. Tout d'abord nous exposons la méthode CART pour la construction des DT. Ensuite nous expliquons en détail la phase d'apprentissage du modèle dans le cadre de RF. Enfin nous détaillons l'étape de classification qui correspond à l'évaluation d'un nouvel élément.

CART

La méthode CART permet de construire les arbres de décision. L'idée générale est de partitionner récursivement l'ensemble des données de départ de manière binaire. Chaque sous-partition est déterminée de façon optimale par une découpe.

L'équation (5.3) modélise chaque trace comme une liste de caractéristiques couplée à un mot-clé. Ainsi, on note C l'ensemble des vecteurs de caractéristiques et $C \subset \mathbb{R}^p$ avec p le nombre de caractéristiques. On défini la notation c^j pour désigner la j-ème caractéristique du vecteur c avec $j \in \{1, ..., p\}$.
On rappelle également qu'un mot-clé se note m et $m \in \{M_{nl} \cup leg\}$. L'ensemble des données d'apprentissage noté E est définit comme suit : $E = \{(c_1, m_1), ..., (c_n, m_n)\}$, avec $c_i \in C$ et $n = |E|$.

A présent, pour la méthode CART, la première étape consiste à découper l'ensemble des données d'apprentissage E en deux. Pour cela, nous allons opérer une découpe qui partitionnera E en deux. Une découpe est une condition de la forme : $c^j \leq d$ où $d \in \mathbb{R}$. Cela permet de créer deux ensembles :

$$E_1 = \{c \in C \mid c^j \leq d\} \tag{5.5}$$

$$E_2 = \{c \in C \mid c^j > d\} \tag{5.6}$$

$$\text{avec } |E_1| = q \text{ et } |E_2| = n - q$$

La sélection de la découpe devra être optimale et sélectionner le bon couple (j, d) afin de minimiser l'impureté des nouveaux ensembles. Cette impureté est calculée via le critère de Gini [118] que nous notons C_g.

Le critère mesure l'impureté d'un ensemble avec une valeur comprise entre 0 et 1. S'il vaut 0, l'ensemble évalué est pur, tous les vecteurs de caractéristiques dans une même découpe correspondent au même mot-clé. A l'inverse s'il vaut 1, l'ensemble est totalement impur, tous les vecteurs de caractéristiques correspondent à des mots-clés différents. A chaque découpe nous cherchons à faire diminuer l'impureté globale. Pour faire cela nous introduisons ζ tel que :

$$\begin{aligned} \zeta =& C_g(E) - (\frac{q}{n} C_g(E_1) + \frac{n-q}{n} C_g(E_2)) \\ & \text{avec } C_g(E) = 1 - \sum_{i=1}^{m} f_i^2 \end{aligned} \tag{5.7}$$

si f_i désigne la fréquence du label i dans E et m est le nombre de labels différents de E.

Ainsi, pour chaque découpe nous cherchons à maximiser ζ, en diminuant l'impureté des sous-ensembles $C_g(E_1)$ et $C_g(E_2)$ (proportionnellement à leur nombre d'éléments) par rapport à $C_g(E)$ en trouvant le bon couple (j, d).

Finalement, les découpes sont réitérées sur chaque sous ensemble sauf si un des sous-ensembles remplit une des trois conditions d'arrêt suivantes :

— Le sous-ensemble est pur et possède donc, un indice de Gini égal à 0.

— Le sous-ensemble ne peut pas être découpé car le score ζ serait négatif (on rendrait les sous-ensembles moins purs).

— La profondeur maximale de l'arbre décisionnel (max_depth) est atteinte pour le nouveau sous-ensemble. Cette valeur sera fixée lors de l'évaluation de la solution dans la section 6.2.2.

Une fois que tous les sous-ensembles ont atteint une condition d'arrêt, l'arbre de décision est construit.

Afin d'illustrer nos propos, nous proposons dans la figure 5.4 un exemple d'arbre binaire avec les critères de découpe, associés au partitionnement d'un espace, dans le cas simple d'une classification à deux caractéristiques. E correspond à l'ensemble de l'espace (à droite de la figure). On scinde cet ensemble une première fois en deux sous-espaces (E_1 et E_1') grâce à la condition $c^1 \leq d_1$ (la caractéristique 1 doit être inférieure ou égale à $d_1 \in \mathbb{R}$). Cette découpe diminue l'impureté globale en rendant chaque sous-espace plus pur. On cherche ensuite une nouvelle découpe et ainsi de suite jusqu'à arriver à une condition d'arrêt défini plus haut.

Maintenant que la méthode de construction des arbres est définie, nous allons décrire le fonctionnement de l'algorithme RF.

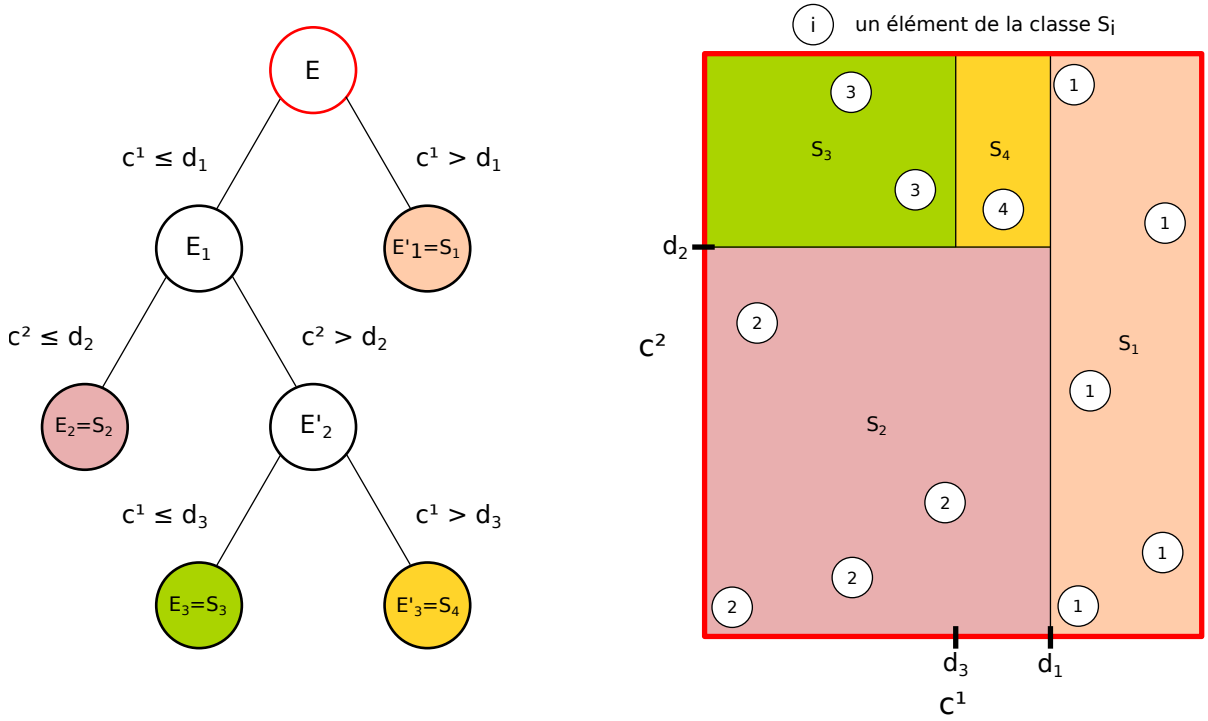


FIGURE 5.4 – Exemple d'utilisation de CART pour un problème à deux variables c^1 et c^2 (d'après R. Genuer [119])

Apprentissage RF

Nous décrivons ici, l'étape (4) d'apprentissage du modèle de la figure 5.2. L'algorithme forêt d'arbres décisionnels étend la méthode CART et rend la classification plus générique en multipliant les arbres de décision. Il utilise la méthode du Bagging (pour Bootstrap et Aggregating) qui se construit en deux étapes : amorçage et agrégation. L'étape d'amorçage concerne l'apprentissage et l'agrégation intervient pour la classification. L'amorçage va créer différents sous-ensembles d'échantillons d'apprentissage. Le nombre de sous ensembles est égal au nombre d'arbres souhaités et est défini par le paramètre n_a. Chacun des n_a sous-ensembles est généré aléatoirement en tirant au hasard (avec remise) l échantillons avec $l < n$, n étant le nombre total d'échantillons. On obtient donc n_a sous-ensembles composés de l échantillons. Ensuite, pour chaque sous-ensemble, un arbre décisionnel est construit avec une version légèrement modifiée de CART : CART RI pour Random Input.

La différence avec CART standard que nous avons décrit précédemment, se trouve au niveau des découpes. Au lieu de chercher une découpe optimale sur l'ensemble des caractéristiques, m_f caractéristiques sont tirées au sort (avec remise) pour chaque découpe ; m_f correspond à la racine carrée du nombre total de caractéristiques.

L'étape d'amorçage et l'usage des entrées aléatoires (RI) pour CART introduisent donc deux sources d'aléa dans la construction de chaque DT ce qui réduit la variance de l'algorithme afin qu'il généralise mieux et évite le surapprentissage. Finalement, l'ensemble des arbres construits constitue le modèle appris par RF et sera utilisé pour la classification. La figure 5.5 schématise l'ensemble des étapes de l'algorithme RF décrites précédemment.

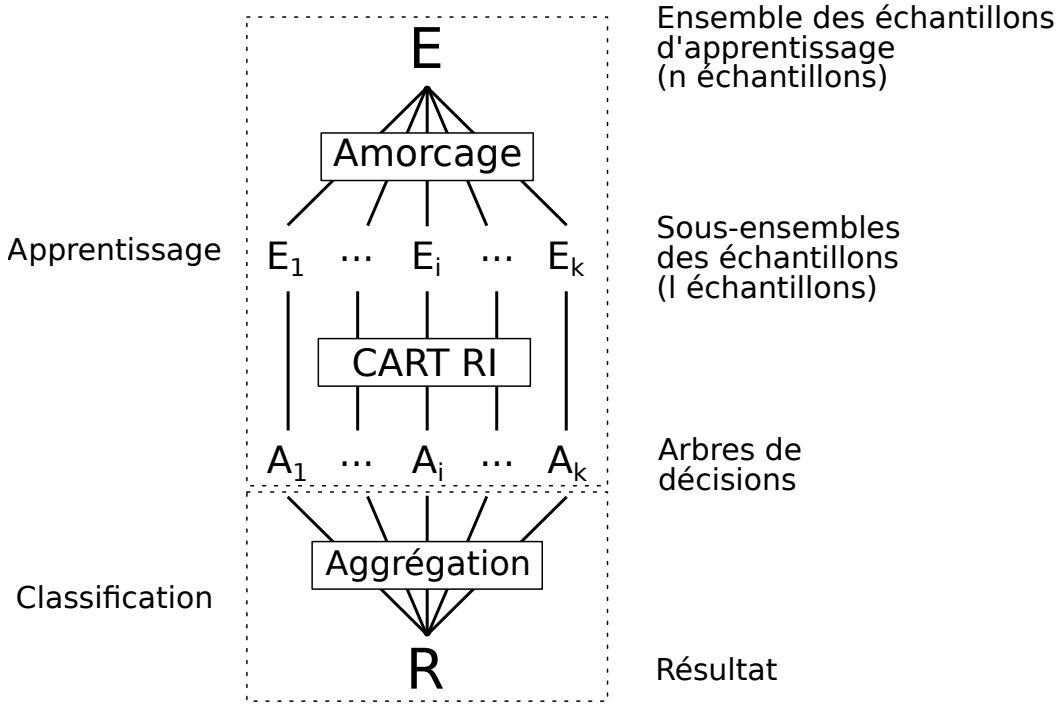


FIGURE 5.5 – Schéma globale de l'algorithme RF

Afin de détecter les traces légitimes nous avons créé une classe légitime nommée *leg* qui est entraînée avec un ensemble de traces issues de différents mots-clés légitimes. Le nombre de traces nécessaires à l'entraînement de cette classe sera déterminée dans le chapitre suivant lors du réglage de l'ensemble des paramètres (cf. section 6.2.2). Cette solution pour la détection des traces légitimes est simple à mettre en place et a déjà été utilisée dans les travaux de Tao Wang *et al.* [120] dans le cadre du website fingerprinting.

Classification RF

La classification fait référence à l'étape (c) du schéma global de la solution (figure 5.2). Avec l'algorithme RF cette étape est très simple. En effet, pour classer un nouvel échantillon (vecteur de caractéristiques), ce dernier est évalué par chaque arbre décisionnel du modèle. Après cela, chacun des arbre donne une prédiction qui correspond à une classe dans l'ensemble $\{M_{nl} \cup leg\}$. Finalement on obtient un ensemble de probabilité pour l'ensemble des classes et l'algorithme désigne la classe la plus probable comme résultat de la prédiction. C'est ce résultat qui sera ainsi retourné par notre solution.

5.4 Synthèse

Dans ce chapitre l'objectif était de définir une nouvelle méthode afin de superviser du trafic HTTP/2. Dans un premier temps, nous avons élaboré un ensemble de caractéristiques capable de distinguer différentes traces utilisant le protocole HTTP/2 sécurisé. Ensuite en combinant ces caractéristiques à de l'apprentissage automatique et en particulier l'apprentissage supervisé avec l'algorithme des forêts d'arbres décisionnels (RF : random forest), nous avons défini une nouvelle

solution de classification.

L'ensemble des caractéristiques que nous avons retenues se compose d'une partie issue de l'état de l'art et d'une deuxième plus adaptée au trafic HTTP/2. Les caractéristiques utilisées sont décorrélées du temps pour minimiser l'impact de l'environnement sur l'identification des traces et sont construites à partir de statistiques issues de l'ensemble des entêtes des records TLS.

La solution de classification est construite comme H1Classifier avec deux phases : la génération du modèle puis la phase de supervision. La génération du modèle se décompose également en plusieurs étapes : la capture des traces d'entraînement, l'extraction des caractéristiques puis l'apprentissage du modèle. Ensuite pour la phase de supervision la solution doit extraire les caractéristiques des traces à tester puis utilise le modèle entraîné pour calculer le résultat. Comme énoncé précédemment l'algorithme de classification sélectionné pour notre solution est RF et a été décrit en détail dans ce chapitre afin d'expliquer la phase d'apprentissage du modèle ainsi que celle de classification.

Chapitre 6

Évaluation de H2Classifier

Sommaire

6.1 Introduction 98
6.2 H2Classifier : paramétrage et évaluation 98
6.2.1 Jeux de données 99
6.2.2 Optimisation des paramètres 101
6.2.3 Évaluation de la qualité de la classification 107
6.3 Impact temporel 113
6.3.1 Jeu de données 113
6.3.2 Performance de H2Classifier dans la durée 114
6.3.3 Apprentissage régulier du modèle 115
6.4 Portabilité de H2Classifier à d'autres services web 118
6.4.1 Jeu de données 118
6.4.2 Expérience et analyse 119
6.5 Impact de l'environnement de capture 121
6.5.1 Jeu de données 121
6.5.2 Expérience et analyse 122
6.6 Synthèse 124

6.1 Introduction

Dans ce chapitre, nous mesurons les performances et la robustesse de notre méthode de classification du trafic HTTP/2 à travers notre implémentation appelée H2Classifier dans la suite du document. Ces mesures consistent en plusieurs évaluations couvrant différentes situations.

Dans un premier temps, nous voulons vérifier si la classification permet de détecter correctement les mots-clés non légitimes et donc la viabilité de notre approche. Nous devons tout d'abord régler les paramètres de H2Classifier en évaluant leur impact afin d'obtenir des résultats optimaux. Nous quantifions ensuite les performances de la classification via trois expériences menées sur plusieurs services très utilisés dans le cadre d'un scénario en monde ouvert.

En plus de l'évaluation de la classification dans des conditions optimales, nous voulons confirmer la généricité de la solution dans des conditions moins favorables (moins de traces d'apprentissage ou plus de mots-clés à détecter). Ces expériences nous permettent alors de mesurer l'efficacité de H2Classifier pour détecter des mots-clés, légitimes ou non, dans différentes conditions.

Nous étendons ensuite les évaluations et testons la robustesse de H2Classifier face à différentes contraintes (scénario en monde fermé) :

— L'impact du temps : à mesure que le temps passe, notre modèle risque de devenir obsolète à cause des changements continus s'opérant sur les contenus supervisés. Ainsi, nous nous posons deux questions : "combien de temps le modèle de classification reste-t-il efficace ?" et "comment prolonger son efficacité ?".

— La portabilité à d'autres services : H2Classifier est étudié dans le reste de nos travaux sur un nombre restreint de services (entre 1 et 5). Nous testons la portabilité de la classification si nous l'appliquons à des nouveaux services. En particulier, nous mesurons les variations de la performance entre les nouveaux services étudiés.

— L'impact de l'environnement de capture : lorsque nous supervisons plusieurs utilisateurs, il est possible de rencontrer des environnements de capture différents. En effet, les utilisateurs ont pu paramétrer leurs machines de façons diverses. Nous évaluons donc l'impact de ces paramètres environnementaux sur les performances de H2Classifier.

6.2 H2Classifier : paramétrage et évaluation

Nous présentons, dans la suite du chapitre, les jeux de données que nous avons construits pour l'évaluation de H2Classifier, dont l'architecture est décrite dans le chapitre précédent (implémenté en langage python avec la bibliothèque scikit-learn [111] pour l'algorithme RF). Nous présentons également les cinq services retenus pour les évaluations, les conditions de capture et finalement les traces générées. Nos jeux de données sont accessibles publiquement à cette URL : `http://betternet.lhs.loria.fr/datasets/h2classifier/`.

Nous décrivons ensuite les paramètres de la classification reposant sur les forêts d'arbres décisionnels. Nous détaillons comment nous avons réglé ces paramètres pour optimiser les performances. L'évaluation de H2Classifier a été réalisée au travers de trois expériences. La première se concentre sur la détection d'un nombre réduit de mots-clés avec un grand nombre de traces disponibles pour l'entraînement. La seconde évalue les limites de la solution en diminuant le nombres de traces disponibles pour l'apprentissage. Enfin, pour la dernière expérience, nous augmenterons le nombre de mots-clés tout en diminuant le nombre des traces disponibles pour l'apprentissage. Ces expériences nous permettent de mesurer l'efficacité de la classification selon

le dimensionnement des données d'apprentissage. Pour les trois expériences nous considérons un scénario en monde ouvert [44].

Pour rappel, toutes les traces que nous utilisons pour l'apprentissage ou pour le test sont de véritables traces réseau collectées entre un navigateur et un serveur. Nous n'avons appliqué aucune sélection ou filtrage de traces par rapport à notre jeu de données, ainsi certaines traces peuvent contenir des erreurs comme dans le cas d'un déploiement réel.

Les métriques utilisées dans cette section sont identiques à celles définies dans le paragraphe 4.3.1 pour HTTP/1.1.

6.2.1 Jeux de données

Présentation des services

Afin de conduire une évaluation plus large que pour H1Classifier, nous couvrons ici plusieurs services pour l'évaluation de H2Classifier. Nous avons ciblé cinq services très utilisés (dans le TOP 30 Alexa [45]) et de natures différentes : moteur de recherche, boutique en ligne, carte interactive, réseau social et moteur de recherche d'images. Le point commun entre ces services est de permettre la recherche de contenu spécifique au moyen d'une requête dans un champ de recherche. Ci-dessous se trouve la liste des services :

— *Amazon* (Alexa rang 14 : `https://www.amazon.com`) : le site de e-commerce le plus visité au monde. Nous nous intéressons au champ principal de recherche qui délivre une page de produits correspondant à une recherche.

— *Instagram* (Alexa rang 29 : `https://www.instagram.com/`) : un réseau social basé sur le partage de photos ou vidéos. Le champ de recherche principal redirige vers des pages de profils.

— *Google Search* (Alexa rang 1 : `https://www.google.com/`) : le moteur de recherche le plus utilisé au monde. Les recherches redirigent vers une liste de pages web et d'informations relatives à la recherche.

— *Google Images* (Alexa rang 1 (google.com) : `https://images.google.com`) : le moteur de recherche d'images de Google. Les recherches sur ce service renvoient des pages d'images associées à la requête.

— *Google Maps* (Alexa rang 1 (google.com) : `https://maps.google.com`) : un service de carte interactive proposé par Google. Le champ de recherche de ce service renvoie une page composée d'un plan de l'emplacement demandé avec ses alentours ainsi que diverses informations.

Les mots-clés recherchés sur les services Google et Google Images sont un sous ensemble des mots-clés du jeu de données $data_2$ précédemment utilisé pour HTTP/1.1. Pour rappel ces mots-clés viennent de titres de pages Wikipedia (en anglais).

Pour le service Google Maps, nous avons retenu la liste des villes françaises [46] couplée avec

44. Rappel : cela signifie que nous souhaitons détecter un certain nombre de mots-clés que nous avons appris à l'avance mais que nous pouvons rencontrer lors de l'évaluation des traces reliées à d'autres mots-clés inconnus (légitimes)

45. `https://www.alexa.com/topsites`

46. `http://www.nosdonnees.fr/wiki/images/b/b5/EUCircos_Regions_departements_circonscriptions_communes_gps.csv.gz`

celle des plus grandes villes du monde [47]. Nous avons procédé a un mélange des deux listes précédentes pour obtenir la liste finale des mots-clés.

Afin de trouver des mots-clés pertinents pour le service Amazon nous avons construit notre liste à l'aide des méta données du jeu de données de J. McAuley [121] qui est composé des commentaires sur différents produits vendus sur Amazon avec le détail de ces derniers. A partir de ces informations nous avons extrait la liste des titres de produits. Le titre étant souvent très complexe, nous avons simulé ce qu'un utilisateur chercherait en sélectionnant les trigrammes de mots-clés les plus fréquemment rencontrés dans la liste des titres de produits.

Enfin, pour le service Instagram nous avons composé la liste de mots-clés de noms de profils. La liste des profils provient du site web d'Instagram lui-même [48]. Cependant une partie de ces profils n'étant pas accessible publiquement ou sans contenu nous avons dû les écarter. Finalement, la taille de la liste des noms de profils disponibles pour le service Instagram est de 1662. Avec ce faible nombre de données (comparé aux 22 000 mots-clés pour les autres services) nous ne sommes pas en mesure d'évaluer Instagram avec un scénario en monde ouvert (classification entre traces légitimes ou non légitimes). Ainsi dans le suite, le service Instagram sera traité indépendamment avec un scénario en monde fermé. Afin de rendre les explications plus simples par la suite, nous utiliserons le terme mot-clé pour les noms de profils également.

Capture des traces

Afin de collecter des traces réseau pour les cinq services cités précédemment, nous avons repris le même procédé que pour H1Classifier exposé à la section 4.2.1. Cependant pour accroître l'efficacité de la phase de capture, nous avons utilisé plusieurs machines virtuelles en parallèle. Chacune d'elle avait sa propre adresse IP publique et était configurée de façon identique, comme décrit ci-dessous :

— Système : Ubuntu 16.04 (VM avec 2 CPU et 4Go de mémoire),

— Navigateur web : Firefox v.63.0 (64-bit),

— Paramètres navigateur : fenêtre en plein écran avec une résolution de 1920×1080, cache désactivé,

— Services : version anglaise des différents sites, paramètres par défaut sur les services.

Nous avons construit deux jeux de données (*data_h2_500* et *data_h2_2000*) dont nous détaillerons ci-après la composition. Le trafic pour les deux jeux de données a respectivement été capturé sur la période du 17 au 25 septembre 2018 pour *data_h2_2000* et du 22 au 28 mars 2019 pour *data_h2_2000*.

Table 6.1 – Jeu de données HTTP/2 : data_h2_2000

Services	Origine	Nombre de mots-clés non légitime/légitime	Nombre de capture par mots-clés non légitime/légitime
Amazon	J. McAuley [121]	2000 / 20 000	12 / 1
Google	Titres de pages Wikipedia ($data_2$)		
Google Images			
Google Maps	Villes françaises ou internationales		
Instagram	Nom de profiles	1662 / n.a.	12 / n.a.

47. `https://datahub.io/core/world-cities`, en ligne en janvier 2020
48. `https://www.instagram.com/directory/profiles/`

Le jeu de données *data_h2_2000* (cf. tableau 6.1) est composé de 12 traces pour chacun des 2000 mots-clés non légitimes associés à chaque service, sauf Instagram. Cela fait un total de 24 000 traces non légitimes par service, soit 96 000 traces au total pour les quatre services : Amazon, Google, Google Images et Google Maps. A cela s'ajoute plus de 20 000 traces correspondant à la capture de mots-clés légitimes ; il n'y a qu'une trace pour chacun de ces mots-clés et ils sont tous différents des mots-clés non légitimes. En ayant à la fois des traces légitimes et non légitimes, cela nous permet de nous placer dans le cas d'un scénario dit en monde ouvert.

Pour le service Instagram nous possédons 12 traces pour 1662 mots-clés non légitimes et aucune trace pour des mots-clés légitimes comme expliqué précédemment.

TABLE 6.2 – Jeu de données HTTP/2 : data_h2_500

Services	Origine	Nombre de mots-clés non légitimes	Nombre de captures par mots-clés non légitimes
Amazon	J. McAuley [121]	500	60
Google	Titres de pages Wikipedia ($data_2$)		
Google Images			
Google Maps	Villes françaises ou internationales		

Le jeu de données *data_h2_500* (cf. tableau 6.2) est composé de moins de mots-clés mais avec plus de traces pour chacun. En effet ce jeu de données comprend 500 mots-clés pour chacun des quatre services considérés (Amazon, Google, Google Images et Google Maps). Chaque mot-clé de chaque service a été capturé 60 fois. Cela fait un total de 30 000 traces par service et 120 000 traces en comptant l'ensemble des quatre services.

Ce jeu de données est complémentaire par rapport au jeu de données data_h2_2000. Il permet de vérifier, à la fois le comportement de notre solution face à plus de variabilité (plus d'échantillons par mot-clé) lors de l'apprentissage et également l'absence de sur-apprentissage (cf. paragraphe 6.2.2).

6.2.2 Optimisation des paramètres

Avant d'évaluer notre classification nous effectuons une phase de réglage des paramètres. Cela permet également de bien comprendre l'impact de ces derniers et donc de mieux les appréhender lors du déploiement de notre solution. Afin de ne pas biaiser l'évaluation avec des paramètres spécifiquement adaptés aux traces testées, nous avons mis de coté un ensemble de traces de test que nous utilisons uniquement pour la partie évaluation (cf. section 6.2.3). Nous avons ainsi mis de coté 4 traces sur les 60 disponible pour les mots-clés non légitimes de *data_h2_500*, ainsi que 15 000 traces légitimes issues de *data_h2_2000*. Il reste donc, 56 traces pour l'ensemble des mots-clés non légitimes issues de *data_h2_500* et 5000 traces légitimes provenant de *data_h2_2000* pour la phase de réglage des paramètres.

Liste des paramètres de l'algorithme à régler :

— **n_estimators** : c'est le nombre d'arbres décisionnels qui doivent composer notre modèle. Plus le nombre d'arbres est élevé plus le modèle est large.

— **max_samples** : c'est le pourcentage maximum de traces considérées pour l'étape de bootstrap. Afin que chaque arbre soit spécialisé, il faut qu'il soit construit différemment des autres. Pour cela nous introduisons de l'aléa via l'étape de bootstrap en sélectionnant

aléatoirement un certain pourcentage des traces d'apprentissage pour chaque arbre. Ce pourcentage correspond à notre paramètre *max_samples*.

— **max_depth** : c'est la profondeur maximale d'un arbre. Lors de la construction des arbres décisionnels avec la méthode CART, la profondeur de l'arbre est une condition d'arrêt.

— **n_feat** : c'est le nombre de caractéristiques effectives que nous gardons pour construire notre modèle. Comme nous l'avons détaillé dans le paragraphe 5.3.2, nous avons un nombre de caractéristiques très important. Ce paramètre réduit le nombre de caractéristiques qui sont effectivement utilisées pour la construction du modèle. Afin de les sélectionner, nous calculons l'importance de chaque caractéristique, comme détaillé par la suite.

— **n_train_leg** : c'est le nombre de traces utilisées pour l'apprentissage de la classe légitime. Chaque mot-clé non légitime sera associé à une classe et l'ensemble des mots-clés légitimes formera une classe.

Méthode d'estimation des paramètres

Les valeurs de paramètres que nous avons retenues ont pour objectif de maximiser les résultats de la classification tout en réduisant la taille du modèle. Plus le modèle a une taille importante, plus il est susceptible d'effectuer du sur-apprentissage car il prend en compte des cas de plus en plus spécifiques. De plus nous cherchons à trouver une *méta-configuration* qui puisse s'appliquer à l'ensemble des services considérés sans faire de concession sur les performances de classification.

Les traces à notre disposition pour le paramétrage sont récapitulées dans le tableau 6.3. 4 traces non légitimes (parmi les 56 disponibles) sont tirées au sort pour chaque mot-clé ainsi que 4600 (parmi les 5000 disponibles) traces légitimes pour construire l'ensemble de test de la phase de réglage. Le reste des traces : 52 traces non légitimes par mot-clé ainsi que 400 traces légitimes serviront pour l'apprentissage.

Table 6.3 – Composition du jeu de données pour la phase de réglage des paramètres (m. = mots-clés et t. = traces)

	Décompte		Description
	Apprentissage	Test	
Service	4		Amazon, Google, Google Images, Google Maps
Traces légitimes	400 m. × 1 t.	4600 m. × 1 t.	Total : 5000 (data_h2_2000)
Traces non légitimes	500 m. × 52 t.	500 m. × 4 t.	Total : 56 (data_h2_500)

Avec cinq paramètres pour quatre services, effectuer plusieurs évaluations indépendantes pour toutes les combinaisons de valeurs de paramètres, impliquerait une grande combinatoire (plus de 5,1 millions de tests) pour 5 évaluations. Sachant qu'il faut plusieurs dizaines de secondes pour chaque test, une recherche exhaustive de toutes les combinaisons de paramètres ne passe pas à l'échelle. Ainsi, nous avons donc évalué successivement les paramètres par sous-ensemble, pour réduire le nombre de tests.

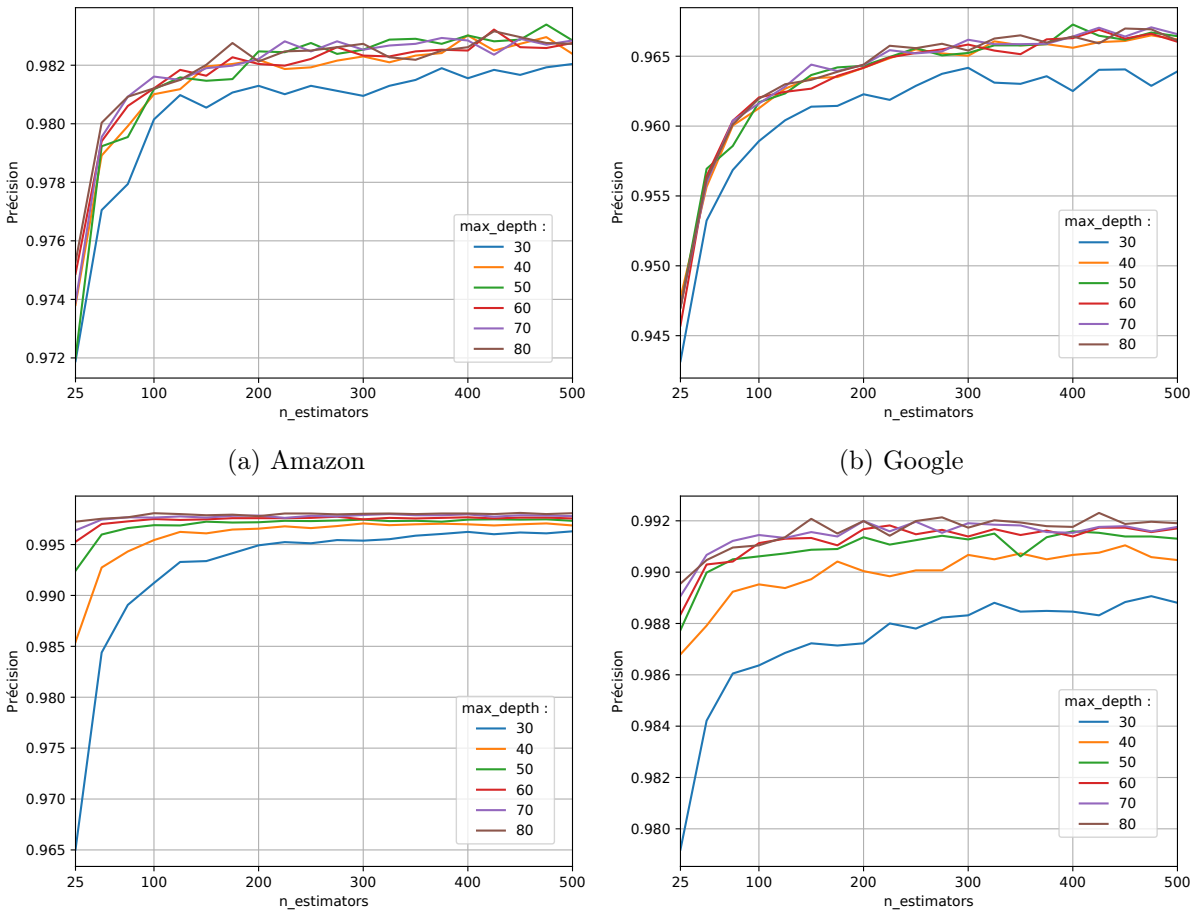
Les paramètres max_depth et $n_estimators$ sont tout d'abord évalués ensemble. En effet ces deux paramètres influencent beaucoup la taille du modèle et nous ne pouvons pas véritablement justifier de valeur par défaut. Pour les paramètres n_feats et n_train_leg nous pouvons affirmer que plus la valeur de ces paramètres est grande plus la classification est simplifiée car nous apportons plus d'informations. Ainsi nous avons défini comme valeur par défaut : $n_feats = 500$

et $n_train_leg = 300$. Cependant nous avons veillé à les réduire par la suite pour diminuer l'impact sur la taille du modèle. Pour le paramètre max_samples, nous choisissons la valeur 0,6 légèrement supérieure à la moyenne afin que chaque arbre soit construit avec une partie commune.

En résumé, nous avons évalué en premier les paramètres max_depth et $n_estimators$ puis n_feats et n_train_leg et enfin $max_samples$.

Dans la suite nous ne présentons que les résultats les plus significatifs ; l'ensemble des résultats est disponible dans l'annexe 6.6.

n_estimators et max_depth :



(a) Amazon

(b) Google

(c) Google Images

(d) Google Maps

FIGURE 6.1 – TPR pour l'évaluation des paramètres $n_estimators$ et max_depth

Pour cette première étape nous faisons varier le nombre d'arbres ($n_estimators$) de 25 à 500 par pas de 25 et la profondeur des arbres de décisions (max_depth) de 30 à 80 par pas de 10. Les valeurs des paramètres n_feats, n_train_leg et $max_samples$ sont fixés, comme expliqué ci-dessus, respectivement à 500, 300 et 0,6.

Pour l'ensemble des valeurs de max_depth, nous avons observé que le taux de faux négatifs (FNR) et le taux de faux positifs (FPR) sont quasi-nuls. Nous avons donc préféré évaluer en détail le TPR pour les quatre services (cf. figure 6.1).

Sur l'ensemble des graphiques, on constate que plus les paramètres max_depth et $n_estimators$ sont élevés, plus le TPR est élevé. Ce taux croit rapidement au début puis l'accroissement ralentit à mesure que $n_estimators$ augmente (augmentation du nombre d'arbres). L'inflexion débute un peu avant 100 arbres pour l'ensemble des services, puis les résultats se stabilisent pour un nombre d'arbres supérieur à 400.

Concernant la profondeur des arbres, pour l'ensemble des services, les courbes du TPR pour $max_depth = 30$ sont nettement inférieures aux autres. A partir d'une profondeur de 40 pour les services Amazon et Google, l'ensemble des courbes se superposent. Pour Google Maps et Google Images, ce constat arrive pour une valeur de max_depth supérieure ou égale à 50. Ainsi, à partir de 50 il n'y a plus d'augmentation significative des performances de la classification.

Ainsi, pour la suite, nous fixons $n_estimators = 400$ et $max_depth = 50$.

n_feats :

Nous faisons varier le nombre de caractéristiques (n_feats) par trace de 25 à 500 par pas de 25. Cela nous permet d'alléger le modèle en ne gardant que les caractéristiques les plus importantes. Le calcul de l'importance des caractéristiques se fait par rapport au nombre d'occurrences de celles-ci dans l'ensemble des arbres du modèle. Plus une caractéristique est présente, plus elle est importante. Nous effectuons donc un premier apprentissage où nous calculons l'importance de chaque caractéristique puis nous conservons les n_feat caractéristiques les plus importantes (traces d'apprentissages et de test). Suite à cette étape nous effectuons l'apprentissage définitif avec le nouveau sous-ensemble de données et nous effectuons ensuite les tests.

Le taux de faux négatifs (FNR) apparaît globalement faible pour toutes les valeurs de n_feats comme indiqué sur la figure 6.2(b). On constate qu'il diminue pour l'ensemble des services à mesure que le paramètre croit. Cependant, on remarquera que les taux pour le service Google sont moins stables et remontent même un peu entre 350 et 500. Pour autant globalement le FNR reste faible pour tous les services (inférieur à 0.07%) pour une valeur de n_feat supérieure à 200.

Le taux de faux positifs (cf. figure 6.2(c)) décroît rapidement pour les valeurs de n_feats entre 25 et 150 puis il se stabilise ensuite. Le FPR est nul pour tous les services excepté Amazon à partir de $n_feats > 250$. Pour le service amazon, le FPR est légèrement inférieur à 0,2% lorsqu'il se stabilise.

Le taux de vrais positifs (cf. figure 6.2(d)) croit rapidement jusqu'à atteindre une valeur stable pour $n_feats > 150$ concernant les services Amazon, Google et Google Maps. Les résultats du TPR pour Google Image nécessite un n_feats plus grand pour le stabiliser. On considérera que n_feats doit être supérieur à 300. La courbe de la justesse visible dans la figure 6.2(a) confirme ce choix. En effet, la justesse se stabilise juste après $n_feats = 100$ pour l'ensemble des services excepté Google Images qui lui nécessite d'augmenter ce taux jusqu'à 300, ce qui limite la complexité de la génération du modèle tout en gardant de bonnes performances de classification.

Par conséquent pour la suite nous fixons $n_feats = 300$.

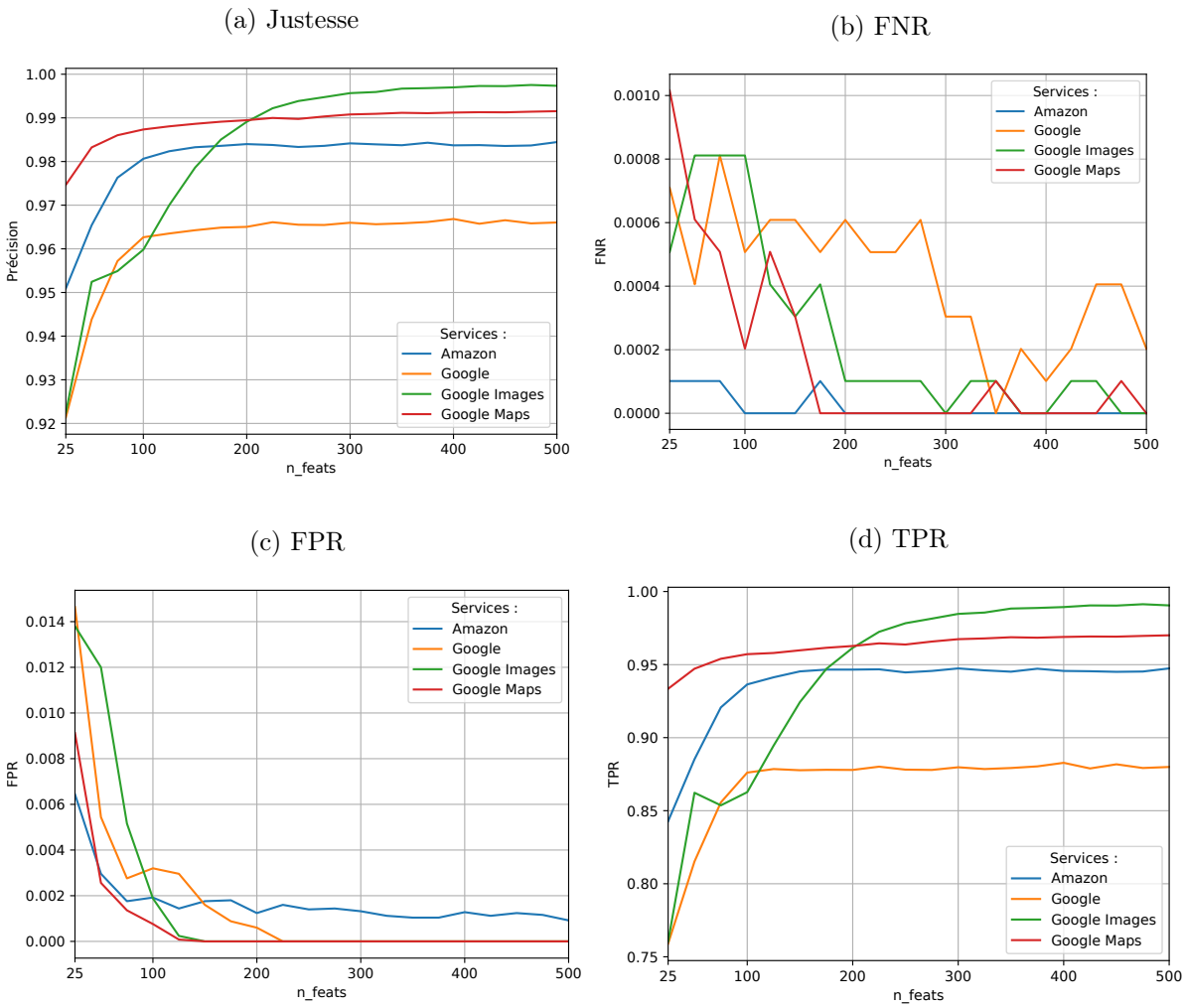


FIGURE 6.2 – Évaluation du paramètre n_feats

n_train_leg :

Pour cette étape nous étudions les résultats de classification lorsque l'on fait varier le nombre de traces légitimes entre 25 et 300 pour l'entraînement. Lors de la phase d'apprentissage, ces traces servent à apprendre le profil de la classe des traces légitimes.

Si le TPR, le FNR et le WCR sont constants pour toutes les valeurs de n_train_leg, le FPR est lui, impacté comme on peut le voir dans la figure 6.3. On constate que le FPR pour les services Amazon et Google est toujours faible (inférieur à 2%). Alors que pour les services Google Maps et Google Images, le paramètre n_train_leg doit être supérieure à 150 pour atteindre cette valeur de FPR.

Ces résultats sont cohérents. En effet, ce paramètre n'affecte que le FPR puisque une trace testée est un faux positif si c'est une trace légitime classée en tant que trace non légitime. Ainsi plus on ajoute de traces pour l'apprentissage de la classe légitime et plus on évite les faux positifs.

En définitive nous fixons $n_train_leg = 150$.

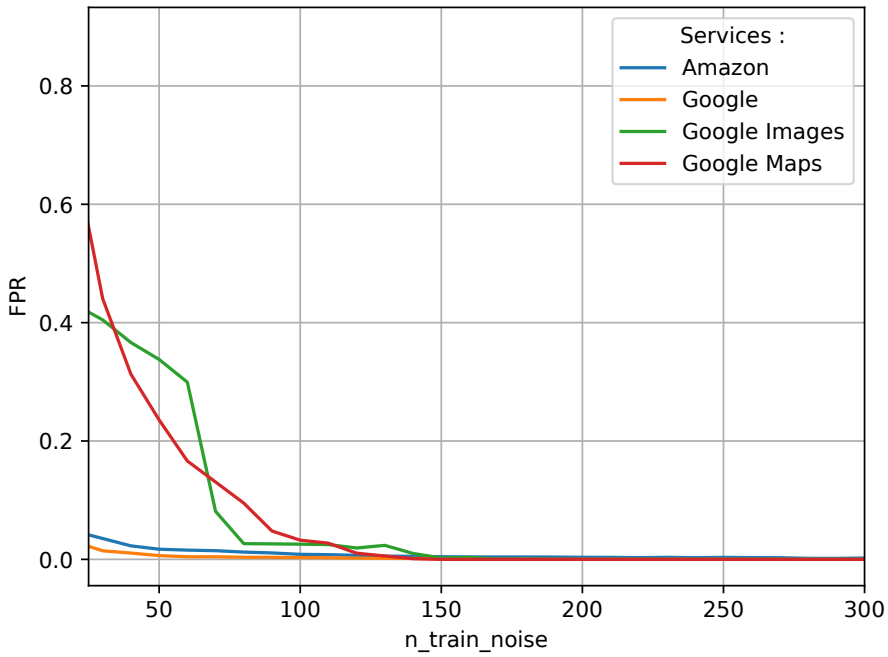


FIGURE 6.3 – FPR relative aux variations du paramètre *n_train_leg*

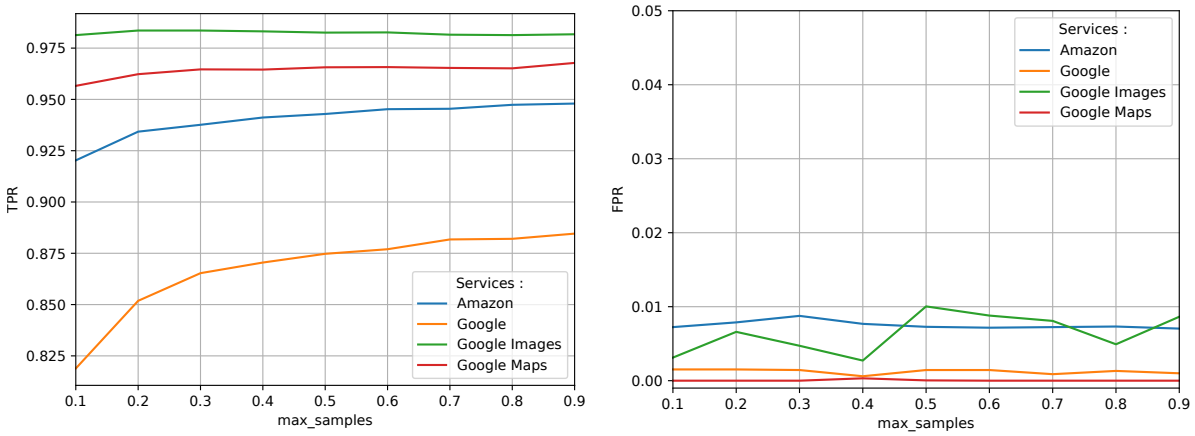
max_samples :

Enfin la dernière étape consiste à faire varier le pourcentage de traces considérées lors de la construction de chaque arbre. Nous avons testé ce paramètre entre les valeurs 0,1 et 0,9 par pas de 0,1. Le cas max_samples = 1 n'est pas considéré afin de toujours construire des arbres différents ni le cas max_samples = 0 qui reviendrait à ne sélectionner aucune trace.

Le taux de faux positif est globalement stable pour les services à l'exception de Google Images. En effet pour ce dernier, la courbe est plutôt fluctuante et a tendance à augmenter. Cependant ces variations sont très modérées car le FPR reste compris entre 0,3% et 1% comme indiqué dans la figure 6.4(b). Le taux de faux négatifs quand à lui reste inférieur à 0,1% pour tous les services.

Pour le TPR (cf. figure 6.4(a)) on constate que plus le paramètre est élevé plus le taux est élevé. Pour les services Google Images le taux est stable pour toutes les valeurs du paramètre. Avec Google Maps et Amazon on observe un léger accroissement. Pour le service Google l'amélioration des résultats est plus importante. Les résultats pour le TPR sont prévisibles car plus le paramètre est grand et plus il y a d'informations pour construire les arbres. Cependant plus *max_samples* est grand, moins il y a d'aléa introduit dans la création des arbres et plus les arbres seront semblables.

Ainsi, pour la suite des travaux nous augmentons légèrement *max_samples* pour atteindre 0,6 car le TPR est amélioré mais nous ne l'augmentons pas davantage afin de garder suffisamment d'aléa. Nous rappelons que l'aléa introduit par ce paramètre (cf. paragraphe 5.3.4) permet à l'algorithme de mieux généraliser et ainsi de limiter les risques de surapprentissage.



(a) TPR

(b) FPR

FIGURE 6.4 – Résultats de l'évaluation du paramètre *max_samples*

Récapitulatifs des réglages

Grâce aux réglages réalisés dans ce paragraphe nous avons pu déterminer les valeurs appropriées pour les paramètres de notre solution H2Classifier :

— **n_estimators** : 400
— **max_depth** : 50
— **n_feats** : 300
— **n_train_leg** : 150
— **max_samples** : 0,6

Ce paramétrage constitue notre *méta-configuration* appliquée dans la suite des travaux. Il peut être adapté au besoin pour être plus spécifique à un service. En effet, nous avons constaté que les paramètres pouvaient avoir des impacts différents en fonction des services. Cependant nous en avons tenu compte dans notre réglage de façon à être le plus générique possible quitte à légèrement accroître la taille du modèle.

6.2.3 Évaluation de la qualité de la classification

Pour rappel, nous évaluons ici les capacités de détection de H2Classifier, en particulier la qualité de ses classifications en fonction du nombre de mots-clés à superviser et du nombre de traces d'apprentissages disponibles pour ces derniers.

Dans la suite, nous utilisons l'ensemble des métriques précédemment définies dans la section 4.3.1. Nous rappelons que ces métriques ont les propriétés suivantes : $TPR+FNR+WCR = 1$ et $TNR + FPR = 1$.

Expérience 1 : nombre de mots-clés réduit et usage d'un large nombre de traces d'apprentissage

Cette première expérience analyse les capacités de détection de H2Classifier avec une faible quantité de mots-clés non légitimes et une grande quantité de traces d'apprentissage pour chacun de ces mots-clés.

TABLE 6.4 – Traces utilisées par service pour l'évaluation 1

	Traces non légitimes (*data_h2_500*)	Traces légitimes (*data_h2_2000*)
Apprentissage	**56** pour chacun des **500** mots-clés	150 parmi 5000
Test	4 pour chacun des **500** mots-clés	15 000

Nous avons utilisé pour chacun des 500 mots-clés de *data_h2_500*, l'ensemble des 56 traces d'entraînement afin de réaliser l'apprentissage du modèle pour les mots-clés non légitimes. Pour chacun de ces 500 mots-clés nous récupérons ici les quatre traces que nous avions préservées pour la phase d'évaluation, comme expliqué à la fin du paragraphe 6.2.2.

Pour les traces légitimes nous utilisons les 15 000 traces de *data_h2_2000* légitimes préservés pour former les traces légitimes de test. Afin de former l'ensemble des 150 traces légitimes nécessaire (n_train_leg = 150, cf. paragraphe 6.2.2) pour l'apprentissage de la classe légitime (*leg*) nous sélectionnons aléatoirement ces dernières parmi les 5000 traces légitimes d'apprentissage. Le récapitulatif de la composition du jeu de données est rappelé dans le tableau 6.4.

TABLE 6.5 – Évaluation H2Classifier avec 500 mots-clés et 56 traces d'apprentissages par mot-clé (moyenne des 5 évaluations et écart-type entre parenthèses)

Source	Amazon	Google	Google Images	Google Maps
TPR	0,946 (2.10^{-3})	0,878 (1.10^{-3})	0,978 (1.10^{-3})	0,971 (1.10^{-3})
TNR	0,997 (4.10^{-4})	0,999 (3.10^{-4})	0,996 (3.10^{-3})	0,999 (4.10^{-4})
FPR	0,003 (4.10^{-4})	0,001 (3.10^{-4})	0,004 (3.10^{-3})	4.10^{-4} (4.10^{-4})
FNR	0 (0)	0 (0)	0 (0)	0 (0)
WCR	0,054 (2.10^{-3})	0,122 (1.10^{-3})	0,022 (1.10^{-3})	0,029 (1.10^{-3})
Justesse	0,991 (4.10^{-4})	0,984 (3.10^{-4})	0,994 (3.10^{-3})	0,996 (3.10^{-4})

Légende : 0 0,25 0,5 0,75 1 (TPR, TNR, Justesse) (FPR, FNR, WCR)

Les résultats obtenus avec H2Classifier pour classifier les traces précédemment énoncées sont rapportées dans le tableau 6.5. Ces résultats sont les moyennes et écarts-types obtenus après cinq évaluations indépendantes.

A partir de ces résultats, on voit que le taux de faux négatifs (FNR) est toujours nul quelque soit le service. Le fait d'avoir un FNR nul, indique que notre classification ne manque aucune occurrence de traces non légitimes dans le cadre de notre expérience.

Concernant le taux de faux positifs (FPR), tous les résultats sont compris entre 0.01% et et 0.4%. Pour la classification, cela signifie que si une trace est reconnue comme étant non légitime, H2Classifier commet une erreur dans moins de 0.4% des cas.

Au vu des résultats pour ces deux dernières métriques (FPR et FNR) qui ont des taux très faibles ou nuls, notre solution est très robuste pour séparer les traces qui correspondent à un mot-clé légitime de celles associés à un mot-clé non légitime.

A présent, nous allons nous intéresser aux taux de vrais positifs (TPR) et de mauvaise clas-

sification (WCR). Comme FNR est nul, le TPR et le WCR sont complémentaires à 1. On voir que le taux de mauvaise classification est globalement plutôt faible : entre 2,2% et 12,2%. Le service le plus favorable pour la classification est Google Images suivi de très près par Google Maps. Ensuite nous avons Amazon avec un peu plus de 5% de mauvaise classification et enfin le moteur de recherche Google. Avec un taux de 12%, ce service est celui dont H2Classifier a le plus de mal à discriminer les traces des mots-clés non légitimes. Cela peut s'expliquer par rapport à la quantité de données disponible sur la page de ce service par rapport aux autres. Par exemple pour le service Google, il y a principalement du texte, ce qui a peu d'impact du la quantité de données envoyée par rapport à des images.

Les résultats varient en fonction du service que l'on cherche à superviser, mais H2Classifier reste très performant dans ce cas d'étude où nous avons peu de mots-clés (500) et beaucoup de traces (56) comme nous le confirme les taux de justesse compris en entre 98,4% et 99,6%.

De manière générale, notre approche est performante dans le cas d'une classification binaire.

Expérience 2 : réduction du nombre de traces d'apprentissages

Table 6.6 – Composition par service pour l'expérience 2

	Traces non légitimes (*data_h2_500*)	Traces légitimes (*data_h2_2000*)
Apprentissage	**11** parmi 56 pour chacun des **500** mots-clés	150 parmi 5000
Test	4 pour chacun des **500** mots-clés	15 000

A présent nous utilisons uniquement 11 traces, dans l'ensemble des 56 traces d'entraînement utilisées, pour l'étape d'apprentissage des mots-clés non légitimes. Le nombre de mots-clés est toujours de 500 et nous utilisons encore pour le test, les quatre traces que nous avions préservées.

Concernant les traces légitimes nous utilisons une nouvelle fois les 15 000 traces légitimes pour les tests et 150 traces légitimes sont sélectionnées aléatoirement parmi les 5000 traces d'apprentissage disponibles. Le récapitulatif de la composition du jeu de données est rappelé dans le tableau 6.6.

Les résultats obtenus avec H2Classifier pour classifier cet ensemble de traces avec un nombre réduit de traces d'apprentissage par mot-clé, sont inscrits dans le tableau 6.7. Ces résultats sont les moyennes obtenues après cinq évaluations indépendantes.

Nous constatons lors de cette deuxième expérience une diminution du TPR pour tous les services. Le taux diminue de 5 points pour le service Amazon, un peu moins de 10 pour Google et environ 1 pour Google Images ou Google Maps. Le FNR étant toujours très faible ou nul (maximum 0.05% pour le service Amazon) cela signifie que la baisse du TPR est essentiellement due à une augmentation du taux de mauvaise classification (WCR). Ainsi, la réduction du nombre de traces d'apprentissage rend la classification entre les mots-clés supervisés plus complexe.

Pour le FPR, le taux est nul pour l'ensemble des service ou extrêmement faible dans le cas de Google images (0,005%). On constate donc que la réduction du nombre de traces d'apprentissage diminue légèrement le FPR en comparaison de l'expérience précédente même si les valeurs étaient déjà très faibles.

TABLE 6.7 – Évaluation H2Classifer avec 500 mots-clés
et 11 traces d'apprentissages par mot-clé
(moyenne des 5 évaluations et écart-type entre parenthèses)

Source	Amazon	Google	Google Images	Google Maps
TPR	0,896 (4.10^{-3})	0,781 (4.10^{-3})	0,969 (1.10^{-3})	0,96 (1.10^{-3})
TNR	1 (0)	1 (0)	0,999 (3.10^{-5})	1 (0)
FPR	0 (0)	0 (0)	2.10^{-5} (3.10^{-5})	0 (0)
FNR	5.10^{-4} (0)	4.10^{-4} (2.10^{-4})	3.10^{-4} (4.10^{-4})	0 (0)
WCR	0,104 (4.10^{-3})	0,218 (4.10^{-3})	0,031 (1.10^{-3})	0,04 (1.10^{-3})
Justesse	0,988 (5.10^{-4})	0,974 (5.10^{-4})	0,996 (1.10^{-4})	0,995 (1.10^{-4})

Légende : 0 0,25 0,5 0,75 1 (TPR, TNR, Justesse) (FPR, FNR, WCR)

Finalement, la réduction du nombre de traces d'apprentissages n'altère pas la capacité de H2Classifier à distinguer un mot-clé non légitime d'un mot-clé légitime, c'est même le contraire (très légère amélioration de cette métrique). Cependant, la réduction du nombre de traces a un impact plus visible sur la capacité de la classification à associer une trace de test au bon mot-clé lorsque qu'elle a été perçue comme non légitime.

Pour la prochaine expérience nous allons augmenter le nombre de mots-clés supervisés tout en conservant un nombre réduit de traces d'apprentissage.

Expérience 3 : augmentation du nombre de mots-clés et réduction des traces d'apprentissage

TABLE 6.8 – Composition par service pour l'expérience 3 (uniquement *data_h2_2000*)

	Traces non légitimes	Traces légitimes
Apprentissage	**11** traces pour chacun des **2000** mots-clés	150 parmi 5000
Test	1 trace pour chacun des **2000** mots-clés	15 000

Cette dernière expérience s'articule autour d'un nombre de mots-clés non légitimes plus important (2000) et un nombre toujours limité de 11 traces d'apprentissage. En effet cette évaluation utilise la totalité du jeu de données *data_h2_2000*. Pour l'apprentissage des classes non légitimes du modèle nous utilisons 11 traces pour chacun des 2000 mots-clés disponibles. Ensuite nous sélectionnons 150 traces aléatoirement dans l'ensemble des 5000 traces d'entraînement disponibles pour l'apprentissage de la classe des mots-clés légitimes.

Pour le jeu de données de test nous utilisons 1 trace pour chacun des 2000 mots-clés non légitimes. La trace est prise au hasard parmi l'ensemble des 12 traces. C'est pour cela que nous avons 11 traces d'entraînement. Concernant les mots-clés légitimes nous utilisons les 15 000 traces de tests.

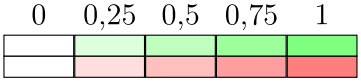
Le récapitulatif de la composition du jeu de données est rappelé dans le tableau 6.8.

Les résultats obtenus pour H2Classifier avec ce nouvel ensemble de traces, sont présentés dans le tableau 6.9. Encore une fois, les résultats sont les moyennes obtenues après cinq évaluations

indépendantes.

TABLE 6.9 – Évaluation H2Classifier avec 2000 mots-clés
et 11 traces d'apprentissages par mot-clé
(moyenne des 5 évaluations et écart-type entre parenthèses)

Source	Amazon	Google	Google Images	Google Maps
TPR	0,81 (4.10^{-3})	0,616 (4.10^{-3})	0,982 (2.10^{-3})	0,913 (2.10^{-3})
TNR	0,996 (5.10^{-5})	1 (0)	0,999 (4.10^{-5})	1 (0)
FPR	0,004 (5.10^{-5})	0 (0)	7.10^{-5} (4.10^{-5})	0 (0)
FNR	6.10^{-4} (4.10^{-4})	4.10^{-4} (6.10^{-4})	8.10^{-4} (6.10^{-4})	0 (0)
WCR	0,189 (4.10^{-3})	0,384 (7.10^{-3})	0,018 (3.10^{-3})	0,087 (2.10^{-3})
Justesse	0,975 (5.10^{-4})	0,955 (9.10^{-4})	0,998 (3.10^{-4})	0,99 (2.10^{-4})

Légende : 

Premièrement, le taux de faux négatifs (FNR), n'a presque pas changé par rapport à l'expérience précédente. En effet, on constate juste une très faible augmentation de 4.10^{-4} à 8.10^{-4} pour le service Google Images, autrement le taux est stable et très faible ($\approx 10^{-4}$). Le TNR est encore une fois très proche de 1 pour l'ensemble des services. Le taux de faux positifs (FPR) est lui aussi extrêmement faible. Pour cette troisième expérience, il est nul pour pour services Google et Google Maps et inférieur à 0,5% pour Amazon et Google Images. Ce taux, comme les deux précédents, ne sont a priori pas impactés par les modifications que nous avons opérées entre les expériences.

Les changements les plus importants sont ceux qui touchent le TPR et le WCR. L'ordre des meilleurs performances de classification en fonction des services n'est cependant pas modifié (Google Images, Google Maps, Amazon puis Google). Globalement les résultats pour ces métriques sont légèrement moins bons que précédemment (en moyenne, TPR : -7,1% et WCR : +7,1%), à cause des conditions plus complexes (plus de mots-clés à superviser et moins de traces pour ces mots-clés) de cette nouvelle expérience.

Synthèse des trois expériences :

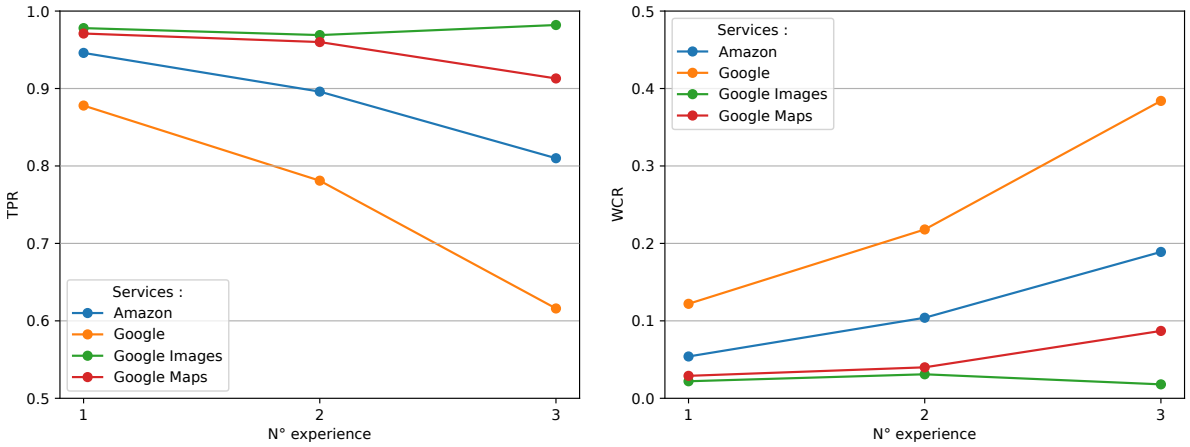
Avant de détailler l'évolution des différentes métriques, nous constatons que les écarts types produits pendant nos expériences, sont toujours très faibles (inférieurs à 7.10^{-3}). L'aléa introduit lors des cinq évaluations pour chaque expérience n'a donc pas d'impact significatif sur les résultats.

Pour l'ensemble des expériences, le FNR est toujours inférieur à 0,1%. Le fait d'avoir un FNR nul ou très faible signifie que notre classification ne manque presque aucune occurrence de traces non légitimes. Lorsque H2Classifier identifie une trace comme étant légitime, il y a une probabilité de moins de 0,1% que la classification soit erronée. Ce résultat est valable avec la réduction du nombre de traces d'apprentissage ainsi qu'avec l'augmentation du nombre de mots-clés pour tous les services.

Le taux de faux positifs est lui aussi extrêmement faible ; il est inférieur ou égal à 0,4% pour l'ensemble des expériences et pour tous les services. Cela signifie que si une trace est reconnue

comme étant non légitime, H2Classifier commet une erreur dans moins de 0,4% des cas.

Au vu des résultats pour ces deux dernières métriques (FPR et FNR) qui ont des taux très faibles, notre solution apparaît très robuste pour identifier une trace légitime. De plus, la réduction du nombre de traces d'apprentissages ou l'augmentation du nombre de mots-clés non légitimes n'impactent pas ce résultat.



(a) TPR (b) WCR

FIGURE 6.5 – Récapitulatif des variations du TPR et WCR en fonction des expériences

Concernant le taux de vrais positifs (TPR) et de mauvaise classification (WCR), la figure 6.5 montre que, pour tous les services excepté Google Images, le taux de mauvaise classification (récip. TPR) augmente (récip. diminue) entre les résultats de l'expérience 1 et 2 puis encore une fois entre ceux de l'expérience 2 et 3. Intuitivement, ce résultat était attendu. En effet, pour l'expérience 2, le nombre de traces d'entraînement est diminué donc l'apprentissage s'effectue avec moins d'échantillons différents. Pour l'expérience 3, en plus du nombre réduit de traces nous avons augmenté le nombre de mots-clés non légitimes (de 500 à 2000). Dans ce cas également il semble naturel que le taux de mauvaises classifications soit plus élevé car la solution de classification a maintenant plus de choix et donc plus de probabilité de classer une trace dans la mauvaise catégorie.

Pour le service Google Images on retrouve une très légère dégradation des résultats, comme pour les autres services, entre les expériences 1 et 2 ; avec cependant une diminution du WCR (récip. une augmentation TPR) pour l'expérience 3 observé pour aucun autre service. La dégradation entre l'expérience 1 et 2 confirme que le nombre de traces pour l'apprentissage joue un rôle sur le taux de bonne classification. Cependant pour l'expérience 3 le résultat est plutôt surprenant et peut s'expliquer par la différence des mots-clés qui sont utilisés entre les expériences. Certain mots-clés sont probablement plus facilement identifiables par rapport à d'autre. Ces variations restent, tout de même, plutôt faibles ; toutes les mesures du WCR sont comprises entre 1,8% et 3,1%. Globalement pour Google Images les résultats se maintiennent pour l'ensemble des expériences.

En plus des résultats du tableau 6.9, nous avons testé H2Classifier sur le service Instagram. Pour cela nous avons réalisé une classification en monde fermé, ce qui signifie que nous n'avons pas

intégré de mots-clés légitimes. Dans ces conditions pour 93% des traces testées H2Classifier était capable de classer ces dernières dans la classe du mot-clé correspondant (**Justesse = 0,93**). Ce résultat n'est pas comparable avec les précédents mais il nous permet d'avoir un premier aperçu de l'applicabilité de H2Classifier sur un service inconnu. Nous testerons plus en profondeur cette possibilité dans la section 6.4.

Pour conclure, les résultats montrent que notre solution H2Classifier est capable de répondre au problème posé illustré sur quatre services très utilisés. Pour ces services, H2Classifier peut détecter si une trace correspond à une liste de mots-clés non légitime ou non dans plus de 99,9% des cas avec un taux de faux positifs inférieur à 0,1%. En plus, nous somme capables de déterminer dans le cadre de la dernière expérience (2000 mots-clés non légitimes avec un nombre de traces restreintes) quel mot-clé est recherché par un utilisateur si ce dernier est non légitime avec un taux de succès qui varie de 61% à 98% en fonction du service.

6.3 Impact temporel

Cette section évalue l'efficacité de H2Classifier dans le temps. Nous introduisons tout d'abord le nouveau jeu de données qui s'étend sur plusieurs mois, utilisé ensuite pour tester H2Classifier dans ces nouvelles conditions. Suite à cela nous proposons un mécanisme de réapprentissage pour maintenir les résultats dans le temps.

6.3.1 Jeu de données

Pour mettre en avant l'impact du temps sur H2Classifier nous avons collecté un ensemble de traces régulièrement pendant une période de quatre mois.

Dans cette expérience, quatre services majeurs ont été étudiés : Amazon, Instagram, Google et Google Images. Nous nous sommes servis d'une liste commune de 500 mots-clés pour les services Amazon, Google et Google Images. Cette liste de mots-clés est un sous ensemble des mots-clés utilisés pour le service Amazon présenté dans le paragraphe 6.2.1. Pour rappel, ce sont des trigrammes de mots extraits de titres de produits vendus sur le site Amazon, provenant du jeu de données de J. McAuley [121]. Pour Instagram nous avons utilisé la liste des 500 profils publiques ayant le plus de "followers". Les pages chargées pour ce service correspondent donc à la page du compte recherché.

Pour chacun des services précédemment cités, nous avons collecté quatre fois par jour le trafic issu du chargement des pages correspondant à 500 mots-clés différents. Afin de capturer les traces, nous avons utilisé la même solution de capture que celle présentée dans le paragraphe 6.2.1. Cependant, cette fois-ci, nous avons en plus effectué une capture d'écran de chaque page capturée pour observer manuellement les évolutions éventuelles du visuel des pages. Ainsi, un total de 8000 traces (4 services $\times$ 4 traces $\times$ 500 mots-clés) quotidiennes ont été collectées. La capture des traces s'est déroulée sur une période de 121 jours entre le 10 mai 2019 et le 15 septembre 2019. Il faut noter que la collecte fut interrompue pendant cinq jours en juin (entre le 21 et le 26) et pendant deux jours en juillet (entre le 21 et le 22) pour des raisons de maintenance de l'infrastructure hébergeant les machines de collecte. Le tableau 6.10 récapitule un ensemble d'informations sur le jeu de données.

Comme pour les précédents jeux de données, les traces sont accessibles publiquement. Elle sont disponibles sur demande et les informations sont sur la page : `http://betternet.lhs.loria.fr/datasets/h2classifier/index.html`.

TABLE 6.10 – Jeu de données temporel

Services	Amazon, Instagram, Google, Google Images
Durée de l'expérience	121 jours
Nombre de mots-clés par service	500
Nombre de traces par mot-clé et par jour	4
Nombre total de traces par jour	8000
Nombre total de traces	968 000
Nombre de postes de capture	1
Taille total du jeu de données	≈ 1,8 To

6.3.2 Performance de H2Classifier dans la durée

Nous mesurons la justesse atteinte par H2Classifier au cours du temps à partir d'un unique apprentissage initial du modèle. Nous considérons ici un scénario en monde fermé pour des raisons pratiques (pas assez de traces dans ce jeu de données pour considérer le cas en monde ouvert). Ainsi nous mesurons ici le cas où H2Classifier doit associer les bons mots-clés aux traces observées.

Nous utilisons H2Classifier avec les paramètres définis dans le paragraphe 6.2.2 et nous apprenons un modèle par service. Pour la phase d'entraînement de chaque modèle, nous utilisons les traces capturées pendant les 15 premiers jours de collecte. Cela correspond à 60 traces d'entraînements par mot-clé (15 jours × 4 traces).

Après la phase d'apprentissage (15 premiers jours), nous avons évalué chaque jour la justesse en utilisant les 2000 traces quotidiennes (4 traces pour chacun des 500 mots-clés) pour chaque service indépendamment. Les résultats de cette expérience sont représentés dans la figure 6.6.

Les périodes grisées sur le graphique correspondent aux périodes de maintenance sans résultat.

La tendance générale est une décroissance de la valeur de la justesse dans le temps pour l'ensemble des services. Cette décroissance des performances dans le temps était attendue avec des résultats qui se dégradent à mesure que l'on s'éloigne de la date initiale d'apprentissage du modèle. En effet, plus le temps passe et plus le contenu associé aux pages supervisées a une forte probabilité d'être mis à jour.

Plus spécifiquement, les résultats pour les services Amazon et Google Images suivent tous les deux une lente et régulière décroissance. On notera tout de même une forte chute des résultats pour le service Amazon aux alentours du 5 septembre. Pour le service Instagram nous observons une décroissance continue de la justesse à l'exception de trois baisses brutales. La première chute se situe autour du 20 juin, elle survient juste avant notre première coupure. Cependant la justesse remonte lors de la reprise des captures le 27 juin. Ensuite nous avons deux autres baisses brutales sans rétablissement de la justesse le 7 août puis le 20 août.

Concernant ces chutes importantes de la justesse constatées pour les services Instagram et Amazon nous présentons ici deux hypothèses. La première est liée à une mise à jour massive de

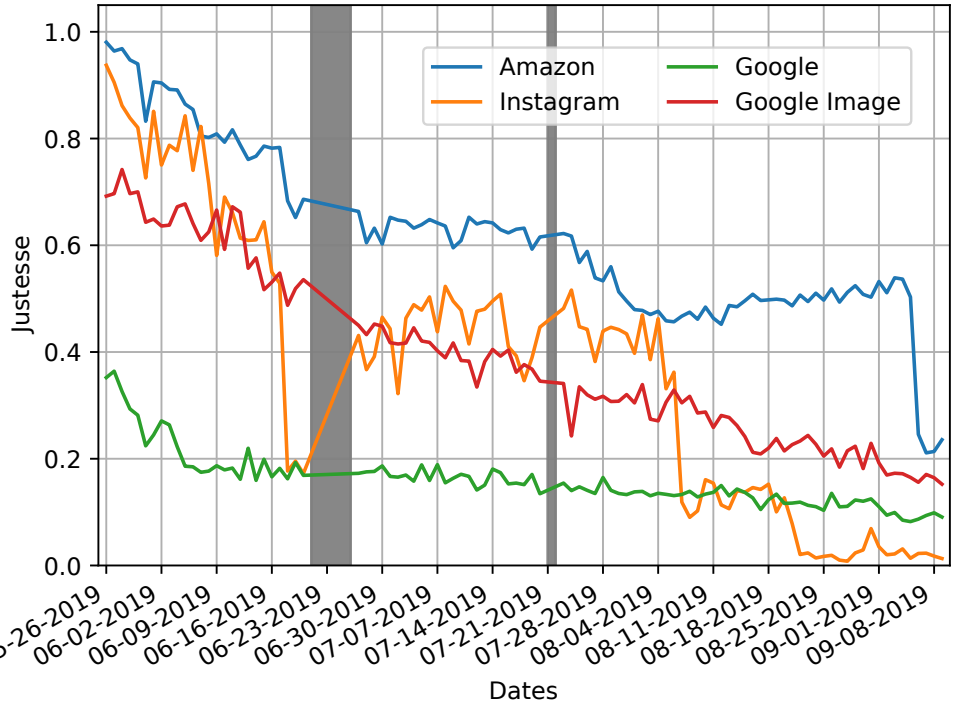


FIGURE 6.6 – Justesse quotidienne moyenne par service

l'ensemble des contenus liés aux pages des mots-clés étudiés. La deuxième implique la manière dont les pages sont envoyées, avec par exemple le changement d'une librairie ou du code de la page sans nécessaire modification du contenu des pages. En observant manuellement une partie des captures d'écrans associées aux pages capturées nous avons pu écarter la première hypothèse. En effet, nous n'avons pas pu établir la présence de changements visuels majeurs entre les captures d'écrans associées aux pages précédant et suivant les différentes baisses brutales du taux de justesse. Ainsi nous pensons que ces chutes de performances sont liées à des mises à jours logicielles (côté serveur) qui ont un impact sur les traces que nous collectons.

Pour le service Google, on observe une décroissance rapide dans les premiers jours avec une valeur initiale de justesse assez faible. Cette faible valeur s'explique car nous construisons une signature sur des captures qui s'étalent sur 15 jours (cf paragraphe 6.3.3 pour les explications). Une fois passée la décroissance des premiers jours, les résultats se stabilisent à hauteur de 20% pour le reste de la durée de l'expérience.

Après le constat de la baisse des capacités de classification à mesure que le temps passe, nous proposons une nouvelle expérience qui vise à tester une solution pour stabiliser les résultats dans le temps.

6.3.3 Apprentissage régulier du modèle

Nous souhaitons observer les performances de H2Classifier lorsque nous réapprenons le modèle à intervalles réguliers afin de stabiliser les résultats. Nous utilisons pour cette expérience les même traces, décrites et utilisées précédemment (cf. paragraphe 6.3.1). Dans cette expérience nous avons appris le modèle tout les n_j jours en utilisant les traces des 15 jours précédant la

date de réapprentissage. Ensuite, nous testons à nouveau la classification avec toutes les traces quotidiennes, associées à l'ensemble des mots-clés. Nous présentons les résultats obtenus pour les valeurs $n_j = 5$ et $n_j = 10$ dans la figure 6.7. Sur cette figure nous avons tracé des lignes pointillées verticales pour indiquer les jours du réapprentissage du modèle. Nous précisons que les périodes de n_j ne prennent pas en compte les jours où la capture était coupée (zones grisées).

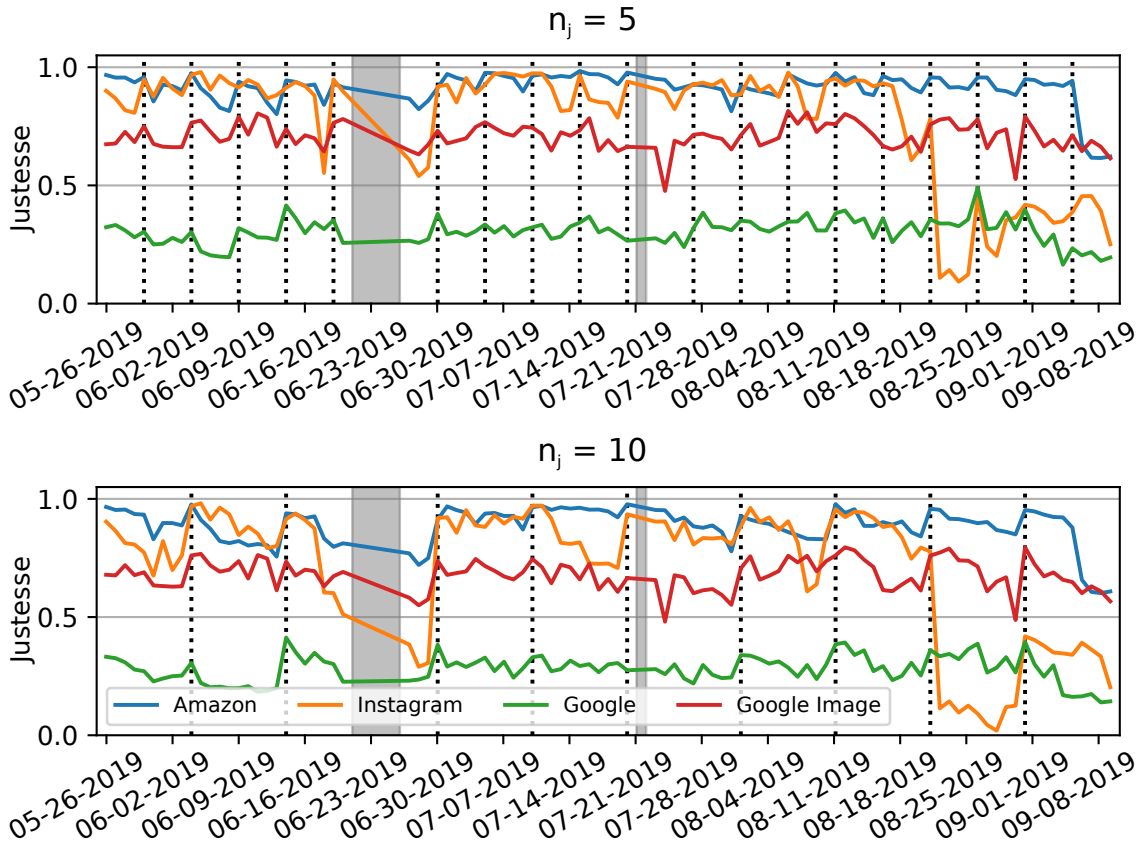


FIGURE 6.7 – Justesse moyenne chaque jour avec apprentissage tous les n_j jours.

Nous observons que dans les deux cas : pour $n_j = 5$ et $n_j = 10$ le réapprentissage du modèle maintient la justesse des premiers jours de test sur l'ensemble de l'expérience. Comme on peut s'y attendre les résultats sont légèrement plus stables pour $n_j = 5$ comme la période de réapprentissage est plus courte. Cependant, dans les deux cas, le réapprentissage stabilise les performances initiales de la classification. Pour les services Google et Google Images dans le cas de $n_j = 5$, les résultats sont stables autour du taux de justesse initial aux alentours de $0,4$ et $0,7$ respectivement. Pour le service Amazon on observe également une courbe plutôt stable. Cependant on voit à nouveau apparaître (comme dans le paragraphe précédent) la baisse de justesse en fin d'expérience (début septembre). Enfin, pour le service Instagram, la chute de la justesse en juin est compensée par le réapprentissage et il en est de même pour le 7 août. Cependant la baisse du 20 août impacte plus durablement les résultats puisque la justesse ne remonte pas complètement après le réapprentissage.

Cette expérience nous permet de mettre en évidence que le réapprentissage régulier, tous les cinq à dix jours par exemple, nous garantit une stabilité des résultats dans le temps. Le temps

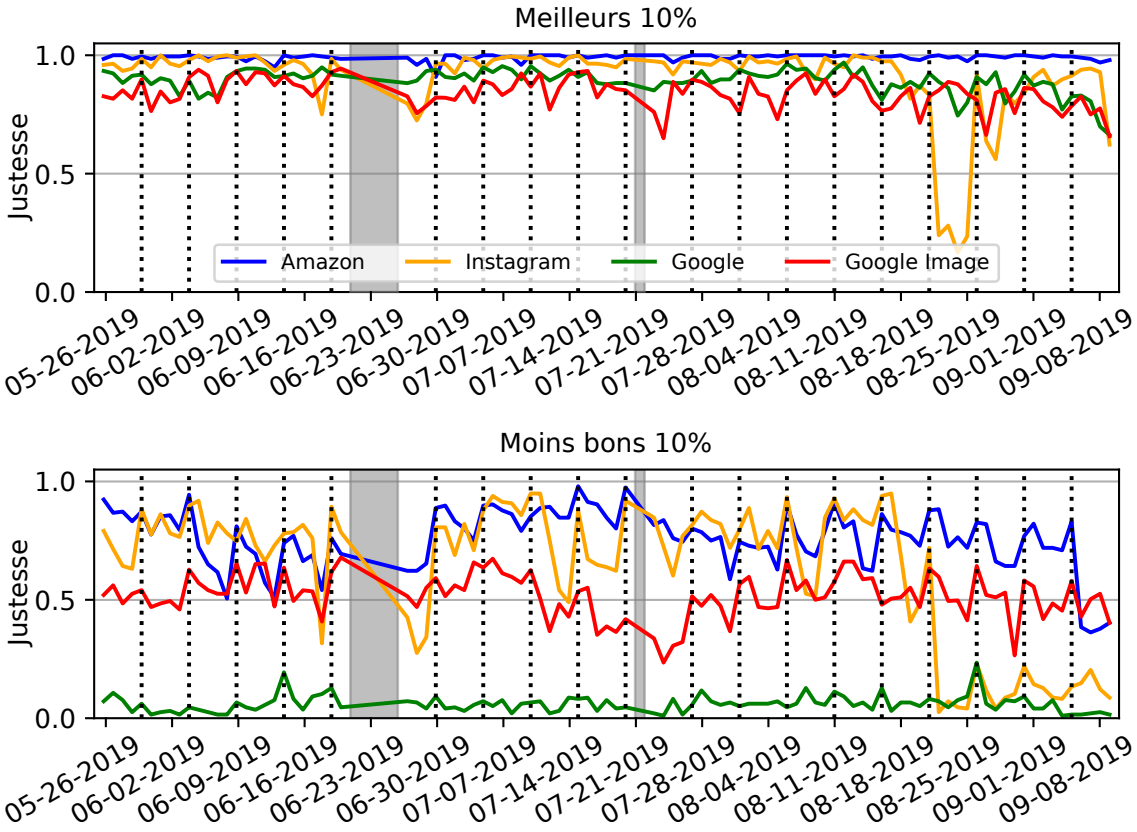


FIGURE 6.8 – Justesse quotidienne moyenne avec apprentissage tout les 5 jours
(en haut : les meilleurs 10% des mots-clés,
en bas : les moins bons 10% des mots-clés)

d'apprentissage du modèle pour un service étant faible (quelques dizaines de secondes), il n'est pas coûteux de mettre en place un tel système si nous pouvons collecter les traces quotidiennement.

Lors de l'observation des résultats au premier jour, nous constatons globalement de moins bons résultats que ceux obtenus dans la section 6.2.3. On passe par exemple de 87,8 % de bonnes classification à 35% pour le service Google. Cela peut s'expliquer à cause de la construction du jeu de données qui est différent sur deux points majeurs :

— L'ensemble des traces capturées pour un même mot-clé est collecté en l'espace de quelques dizaines de minutes dans les expériences précédentes (section 6.2.3, 6.2.3, 6.2.3). A présent, dans cette expérience nous avons un autre scénario et nous apprenons le modèle avec des traces qui sont collectées sur une période s'étendant sur 15 jours. Pendant cette période de temps, la page a évolué. Donc, le modèle est construit à partir de données plus variables, ce qui explique la baisse des résultats sur le service Google. Pour autant, la classification étant entraînée sur des données plus variées, on peut minimiser également le surapprentissage de cette manière.

— Dans cette expérience les mots-clés sélectionnés sont des termes très courants ou populaires (trigrammes les plus fréquents dans des titres d'articles sur Amazon). Dans cette situation les contenus ont plus tendance à être mis à jour fréquemment et donc impactent plus fortement notre classification. On observe de grandes différences de performance de H2Classifier en fonction des mots-clés. Pour illustrer ces propos nous avons tracé sur la

figure 6.8 les résultats en considérant les 10% de mots-clés produisant les moins bons résultats ou à l'inverse, les 10% de mots-clés donnant les meilleurs résultats. La notion de meilleurs ou moins bons 10% est construite en tenant compte de la valeur moyenne de la justesse durant l'ensemble de l'expérience. On constate qu'il y a une très grande disparité dans les résultats et particulièrement pour le service Google.

Ces différences sur les résultats ainsi que les tendances constatées lors de ces expériences nous permettent de mettre en lumière une manière d'évaluer et corriger le problème de l'efficacité des solutions de classification dans le temps en effectuant du réapprentissage régulier. De plus nous avons observé les fortes différences de résultats en fonction des mots-clés considérés ainsi que la période de temps pendant laquelle nous collectons les traces d'entraînement.

6.4 Portabilité de H2Classifier à d'autres services web

6.4.1 Jeu de données

Cette nouvelle expérience cherche à vérifier la généricité de notre approche et donc son applicabilité à un large éventail de services. Le jeu de données dédié pour cette dernière contient des traces relatives à plus de 3000 nouveaux services. La construction s'est déroulée en trois étapes.

Premièrement, nous avons sélectionné les services sur lesquelles nous avons testé H2Classifier, ensuite les mots-clés à rechercher sur ces services et enfin nous avons réalisé les captures de l'ensemble des traces correspondantes. Ces différentes étapes sont détaillées dans la suite de ce paragraphe. Un aperçu du contenu du jeu de données est présenté dans le tableau 6.11.

Table 6.11 – Jeu de données des services

Nombre de services	3096
Nombre de mots-clés par service	20
Nombre de traces par mot-clé	20
Nombre de traces par service	400
Nombre total de traces	1 238 400
Nombre de machines de capture	6
Taille totale du jeu de données	≈ 6 To

Sélection des services

Afin de tester des services communément utilisés nous nous appuyons sur la liste des pages web les plus visitées aux Etats-Unis [49]. Nous excluons automatiquement les services qui ne supportent pas le protocole h2 (HTTP/2 + TLS). Suite à ce premier filtrage, nous testons aussi automatiquement les services afin de détecter s'ils possèdent une barre de recherche. Si c'est le cas, le service est retenu sinon il est mis de côté. Dans certain cas, le handshake TLS s'effectue mais la page ne charge pas ou renvoie une erreur. Ces derniers services sont donc également écartés. Finalement l'ensemble des services retenus est constitué de 3096 éléments.

49. `https://www.quantcast.com/top-sites/US`

Sélection des mots-clés

Pour chaque service considéré, 20 mots-clés différents sont utilisés. Chaque mot-clé étant capturé à 20 reprises, cela fait un total de 400 traces par service comme décrit dans le tableau 6.11. La sélection des mots-clés est importante à cause de la variété des services rencontrés. En effet, la recherche d'un mot-clé pris au hasard n'a pas nécessairement de sens sur un service donné. Cependant, cette sélection doit être automatisée au vu du nombre de services considérés.

Pour essayer d'appliquer des mots-clés en accord avec un service, nous avons classé les services dans différentes catégories. La classification de ces services a été effectuée au moyen de la classification du produit FortiGuard [50]. La figure 6.9 représente la distribution des catégories observées pour les 3096 services retenus. Il y a 61 catégories différentes pour les services considérés, et les catégories avec moins de huit occurrences sont regroupées sous le label "other" dans le graphique. Pour la première douzaine de catégories les plus représentées (74,4% des services) nous avons défini manuellement une liste de mots-clés par catégorie. Pour le reste des catégories nous avons utilisé des mots tirés au hasard dans le dictionnaire (Oxford dictionnary). Ainsi, pour trois quart des services les requêtes sont sémantiquement valides.

Capture des traces

L'ensemble des 20 traces pour les 20 mots-clés de chacun des 3096 services a été capturé entre le 15 juin et le 14 août 2019. L'outil de collecte qui a permis la capture des traces est encore le même que celui présenté dans le paragraphe 6.2.1. Cependant, comme pour l'expérience précédente de la section 6.3 nous avons également effectué une capture d'écran automatique pour chaque page capturée.

La campagne de collecte a été parallélisée grâce à l'usage de plusieurs machines virtuelles avec chacune des adresses IP différentes. Afin de ne pas surcharger les services, nous avons alterné les requêtes par fenêtre de 20 services en collectant une trace toutes les 15 secondes en moyenne. Ainsi, chaque service recevait en moyenne une requête toutes les cinq minutes.

6.4.2 Expérience et analyse

Nous avons entraîné puis testé H2Classifier successivement pour les 3096 services. Parmi les traces de chaque service, nous avons tiré au sort quatre des 20 traces de chaque mot-clé. Ces traces (4 traces $\times$ 20 mots-clés) forment ensemble des traces de test et le reste des traces (16 traces $\times$ 20 mots-clés) forment les traces d'entraînement.

Pour chaque service nous avons ainsi calculé la justesse de notre classification H2Classifier. Les résultats sont tracés sur le graphique de la figure 6.10 qui représente le taux de justesse cumulatif pour l'ensemble des services. Nous avons en ordonnée le pourcentage du nombre de services et en abscisse le taux de justesse. Nous avons tracé avec une droite orange l'efficacité théorique d'une classification aléatoire. Cette droite équivaut à une justesse de 5% (*i.e.* sélection de la bonne classe parmi les 20 classes disponibles).

Sur la figure on peut voir que plus de 50% des services ont un taux de justesse supérieur ou égale à 90%. A l'opposé seulement 10% des services étudiés ont un taux de justesse inférieur à 50%.

Seulement 2% des services testés atteignent un taux de justesse inférieur à 5% qui correspond à un classement aléatoire.

50. `https://fortiguard.com/webfilter`

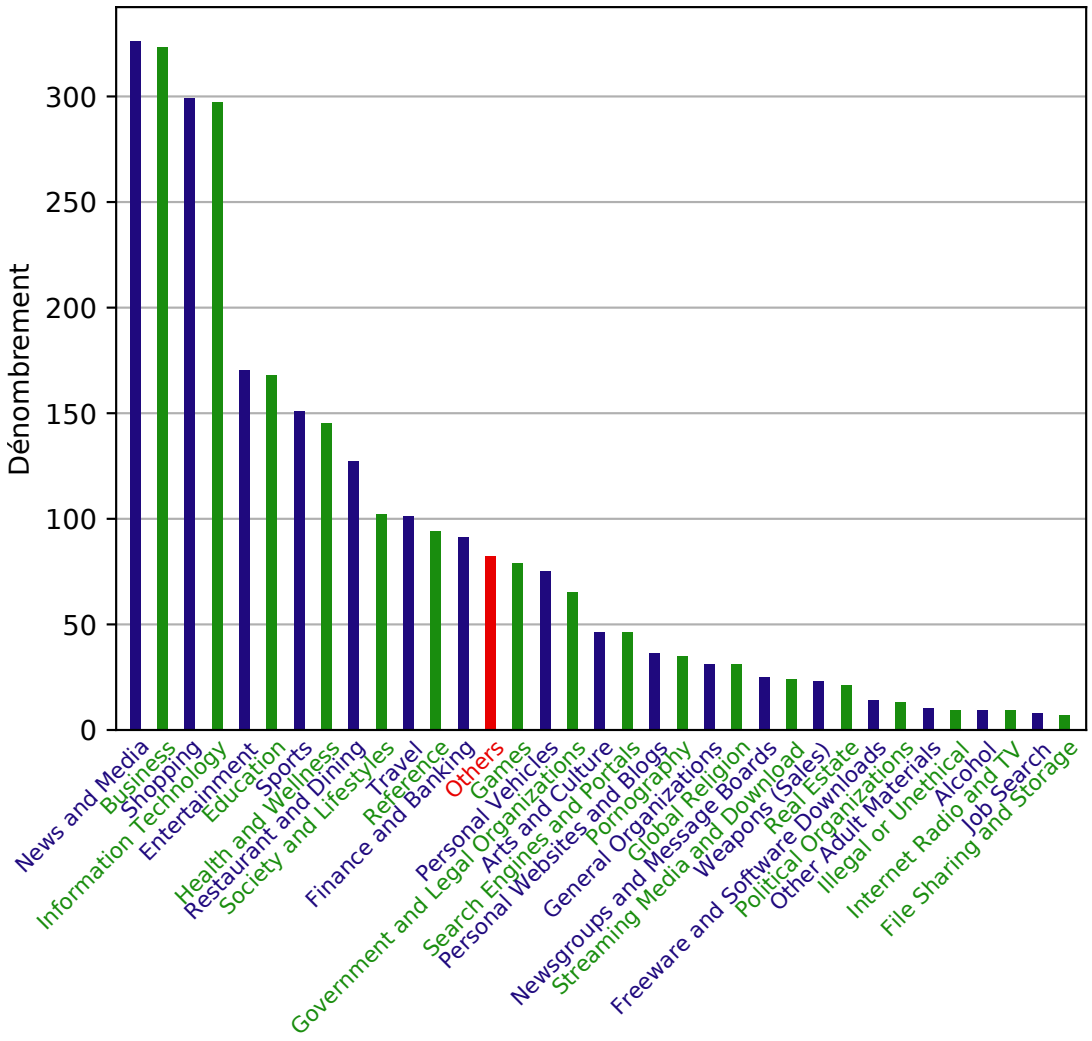


FIGURE 6.9 – Catégories de 3096 services retenues

Nous avons étudié les causes potentielles des mauvaises performances de H2Classifer pour certains services, en analysant manuellement les captures d'écran. Suite à cela, deux causes principales de mauvaise classification se démarquent :

— La page retournée par le service est toujours identique, peu importe le mot-clé. Cela peut être dû à des mots-clés non pertinents pour le service concerné pour lesquels, une page par défaut est retournée. Dans d'autres cas, cela peut être causé par des erreur du serveur (par exemple : maintenance ou erreurs internes du service) qui renvoie encore une fois vers une page par défaut commune à tous les mots-clés.

— La page retournée est bien différente pour chaque mot-clé mais cette dernière est constituée d'un nombre d'objets très réduit. Notre solution H2Classifier a été conçue pour discriminer des pages pour des services complexes composés d'un nombre conséquent d'objets différents. Par exemple, sur le site de news *www.metro.us* le résultat d'une recherche affiche une simple liste de texte sans image. Le faible quantité de contenu dans la page peut expliquer les mauvais résultats pour ce service. Pour la même raison, on a pu observer précédemment que

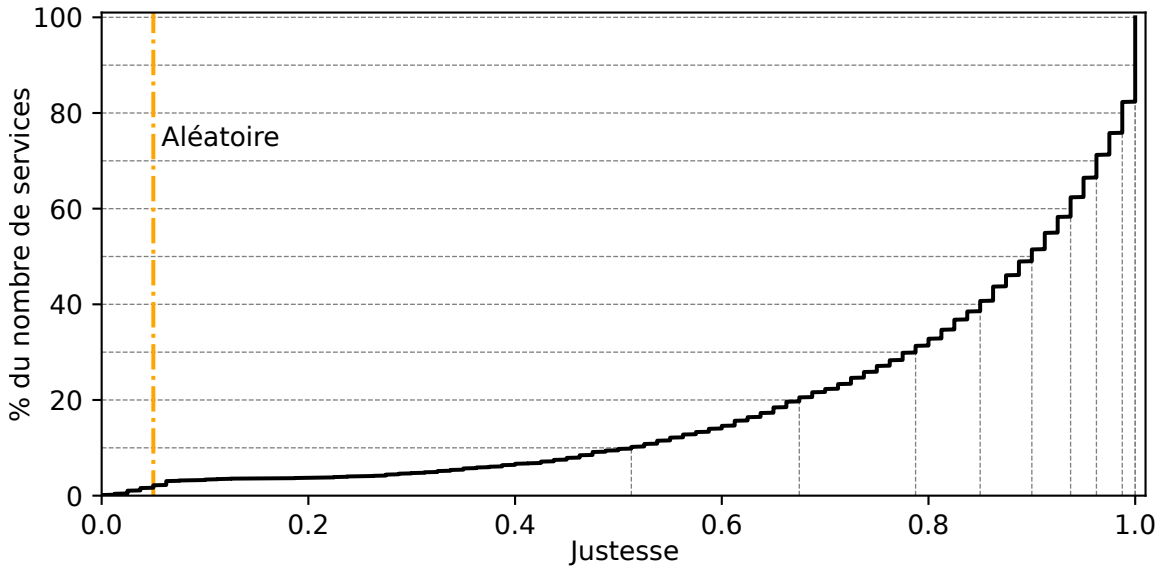


FIGURE 6.10 – Courbe cumulative de la justesse pour les 3096 services testés

les performances de H2Classifer sont meilleures pour le service Google Image que pour le service Google.

Nous rappelons ici, que la sélection des services ne repose pas sur le type de contenu hébergé. Au regard de ces conditions, nous avons tout de même pour plus de 50% des services testés un taux de justesse supérieur à 90%. Notre solution H2Classifier est assez générique pour être utilisée dans la supervision d'un grand nombre de services différents.

6.5 Impact de l'environnement de capture

La configuration du service ainsi que les options du navigateur peuvent affecter le contenu des pages et probablement les capacités de supervision de notre solution. Pour mesurer cela, nous détaillons les configurations que nous avons testées et évaluées.

6.5.1 Jeu de données

Pour cette expérience nous considérons uniquement le service Google Images accédé depuis un navigateur Firefox mais dont la configuration varie comme indiquée ci-dessous, y compris en usurpant le type de navigateur.

Options pour le navigateur Firefox :

- Valeur du User Agent :
  - Firefox (par défaut) : Mozilla/5.0 (X11 ;Ubuntu ;Linux x86_64 ;rv :63.0) Gecko/20100101 Firefox/63.0
  - Android (F1A) : Mozilla/5.0 (Linux ; Android 7.0 ; PLUS Build/NRD90M) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.98 Mobile Safari/537.36

  - Safari (F1B) : Mozilla/5.0 (Macintosh ; Intel Mac OS X 10_14_4) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.1 Safari/605.1.15
- Langue par défaut : Français (par défaut) ou Allemand (F2).
- Résolution de l'écran (taille de la fenêtre du navigateur) : FullHD (par défaut), HD (F3).

Options du service Google Images :

- Safe search : désactivé (par défaut), activé (G1).
- Critère de taille d'images : toutes les images (par défaut), grandes images (G2A) ou petites images (G2B).
- Critère sur la couleur des images : toutes les images (par défaut), images en couleur (G3A) ou images noir et blanc (G3B).
- Critère sur la date : toutes les images (par défaut), ancienneté inférieure à une semaine (G4A) ou ancienneté inférieure à un an (G4B).

Pour cette expérience nous avons capturé 50 traces pour 25 mots-clés différents (noté M_{25}) dans chacune des configurations décrites dans la liste ci-dessus. Les mots-clés de M_{25} ont été sélectionnés aléatoirement parmi l'ensemble des mots-clés utilisés dans le jeu de données *data_h2_500* pour Google Images. Une fois de plus, nous utilisons la même solution de capture décrite dans le paragraphe 6.2.1.

6.5.2 Expérience et analyse

H2Classifier est entraîné pour détecter la configuration par défaut pour 25 mots-clés de M_{25} issus du jeu de données *data_h2_500*. Parmi les 60 traces disponibles pour chaque mot-clé, 50 sont utilisées pour l'apprentissage et 10 pour le test. Ces 10 traces de tests représentent l'échantillon de base par défaut. H2classifier est configuré avec le méta-configuration obtenue dans le paragraphe 6.2.2 après optimisation des paramètres.

Ensuite, pour chacune des traces associées à une nouvelle configuration (25 mots-clés × 50 traces), nous avons effectué la classification de ces dernières et les taux de justesse obtenus sont consignés dans le tableau 6.12.

TABLE 6.12 – Résultats sur l'impact de l'environnement

Options	Par défaut	F1A	F1B	F2	F3	G1
Justesse	1	0,97	1	0,91	0,04	0,85
Options	G2A	G2B	G3A	G3B	G4A	G4B
Justesse	0,29	0,17	0,86	0,18	0,06	0,91

Légende : 0 0,25 0,5 0,75 1

On constate que le test de la configuration par défaut nous donne un taux de justesse de 100%. C'est plus que ce que nous avons obtenu dans l'expérience 1 du paragraphe 6.2.3 mais nous rappelons que nous considérons ici un scénario en monde fermé avec seulement 25 mots-clés différents. Pour les configurations F1A, F1B, qui correspondent à un changement du *User-Agent*, on obtient des résultats quasi-identiques (respectivement 97% et 100%).

Les configurations F2, G1, G3A, et G4B ont des taux de justesse compris entre 91% et 85%. Cela correspond au changement de la langue du navigateur, l'activation de l'option *Safe-Search* ou des filtrages pour conserver uniquement les images de couleur ou datées de la dernière année. Ces configurations ont a priori peu d'impact sur les résultats de la classification. Cela vient du fait que le contenu des pages produites n'a pas beaucoup varié.

Les résultats précédents contrastent beaucoup avec ceux obtenus pour les dernières configurations qui donnent des résultats compris entre 4% et 29%. Les filtres qui conservent uniquement les images en noir et blanc ou les images d'une certaine taille font varier le contenu des pages de manière non négligeable. Ces modifications se font ressentir directement sur les résultats de la classification.

En conclusion pour cette expérience, nous avons pu constater que l'impact de l'environnement de la capture des traces a des répercussions très variables sur la classification selon les configurations choisies.

Dans la suite nous proposons de prolonger cette expérience et d'observer l'efficacité de la classification lorsque l'on utilise quelques traces d'une nouvelle configuration durant la phase d'apprentissage du modèle.

Apprentissage spécialisé

Pour cette nouvelle expérience, nous avons utilisé 40 traces de la configuration par défaut pour l'ensemble des mots-clés de M_{25} avec entre 1 et 10 traces de ce même mot-clé capturé avec une des autres configurations. Nous avons évalué ce modèle avec les 1000 traces restantes (40 traces $\times$ 25 mots-clés). Pour cette expérience nous avons effectué cinq évaluations en prenant à chaque fois un ensemble de traces d'apprentissage différents. Les résultats de cette évaluation sont donnés dans le tableau 6.13.

	Nb. traces de la configuration			
	1	3	5	10
Configuration	Justesse (écart type)			
F1A	0,998 (1.10^{-3})	1 (0)	1 (0)	1 (0)
F1B	1 (0)	1 (0)	1 (0)	1 (0)
F2	0,995 (6.10^{-3})	1 (0)	1 (0)	1 (0)
F3	0,28 (0,043)	0,62 (0,015)	0,85 (0,015)	0,94 (8.10^{-3})
G1	0,87 (7.10^{-3})	0,89 (9.10^{-3})	0,94 (0,019)	0,977 (9.10^{-3})
G2A	0,91 (0,019)	0.99 (9.10^{-3})	0,996 (5.10^{-3})	1 (0)
G2B	0,92 (0,025)	0.998 (1.10^{-3})	0,999 (9.10^{-4})	0,999 (5.10^{-4})
G3A	0,97 (8.10^{-3})	1 (0)	1 (0)	1 (0)
G3B	0,95 (0,013)	1 (0)	1 (0)	1 (0)
G4A	0,62 (0,025)	0,92 (8.10^{-3})	0,96 (0,011)	0,99 (3.10^{-3})
G4B	0,993 (4.10^{-3})	1 (0)	1 (0)	1 (0)

Légende : 0 0,25 0,5 0,75 1

TABLE 6.13 – Résultats sur l'impact de l'environnement en intégrant des traces de la nouvelle configuration durant l'entraînement (moyenne sur 5 expériences)

On constate immédiatement en regardant le tableau que même l'ajout d'une trace d'une nouvelle configuration durant l'entraînement permet d'améliorer considérablement les résultats. Dans les situation F3 et G4A qui affichaient respectivement un taux de justesse de 4% et 6% précédemment, on observe de nouveaux résultats à 28% et 62%. Il également intéressant de noter que plus on ajoute de traces correspondant à une nouvelle configuration et plus les résultats s'améliorent. Lorsque l'on utilise 10 traces issues des nouvelles configurations, on obtient des taux de justesses compris entre 94% et 100%.

En conclusion, l'environnement de la capture peut avoir un impact non négligeable sur les résultats de la classification. En effet, comme nous avons pu le constater certaines configurations réduisent considérablement les performances. Pour améliorer la classification, il est possible d'utiliser des traces provenant de différents environnements pour construire des modèles plus spécifiques. On obtient alors une classification plus robuste à ces changements d'environnement.

Dans le cas où cette solution ne suffirait pas à obtenir les résultats souhaités, une autre solution serait de créer une nouvelle classe dans le modèle avec un nom qui suivrait une nomenclature tel que "<mot-clé>_<configuration>", correspondant à la recherche spécifique d'un mot-clé dans une configuration donnée.

6.6 Synthèse

H2Classifier est l'implémentation de la méthode décrite dans le chapitre 5. L'objectif du chapitre était de présenter son évaluation. Pour cette évaluation, nous avons décidé de réaliser plusieurs expériences qui ont permis de mesurer les performance de notre solution dans différentes situations et donc de vérifier sa viabilité.

Dans la première évaluation, nous avons étudié l'impact des différents paramètres sur les capacités de classification. Après avoir déterminé les paramètres optimaux, nous avons évalué H2Classifier sur quatre services très utilisés. Cette évaluation s'est décomposé en trois expériences avec un nombre de mots-clés à superviser et un nombre de traces d'apprentissage variables. On observe que lorsque l'on souhaite distinguer les traces des mots-clés non légitimes entre elles, globalement les performances baissent si on augmente le nombre de mots-clés à superviser ou si on diminue le nombre de traces d'apprentissage disponibles. Cependant, il est important de noter que notre classification détecte dans plus de 99,9% des cas (FPR inférieur à 0,1) si une trace correspond a une liste de mots-clés légitimes ou non légitimes, quelque soit la situation. En conclusion pour cette évaluation, s'il est nécessaire d'opérer un filtrage strict, la classification binaire est recommandée au vu de ses très bons résultats. Dans le cas de la reconnaissance de la trace de 2000 mots-clés non légitimes parmi un large ensemble de traces légitimes, notre solution affiche un taux de vrais-positifs entre 61% et 98% en fonction du service.

Concernant l'impact temporel sur notre classification, nous avons collecté des traces quotidiennement sur une durée de 4 mois. Avec ce jeu de données nous avons pu constater une décroissance progressive des résultats au cours du temps. Afin de contre balancer cette baisse nous avons mis en place un réapprentissage régulier pour stabiliser les résultats. Nous avons ainsi pu constater une stabilisation des résultats dans le temps en utilisant une fenêtre de réapprentissage de cinq jours. Comme nous l'avons montré, le réapprentissage n'est pas coûteux, ce qui permet à notre approche de rester valable dans le temps.

Ensuite, nous avons présenté notre étude sur la portabilité de H2Classifier pour des nouveaux services. Sur plus de 50 % des 3096 nouveau services évalué, la classification affiche un taux de

justesse supérieur ou égale à 90% et seulement 10% des services ont un taux de justesse inférieur à 50%. En utilisant la méta-configuration, H2Classifier affiche des résultats indiquant une bonne portabilité de la solution pour la supervision de nouveaux services.

Finalement nous avons terminé cette série d'évaluation en mesurant l'impact de l'environnement de capture sur les performance de la classification. Il en ressort que l'environnement de la capture peut avoir un impact non négligeable sur les résultats de la classification et est difficilement prévisible a priori (spécifique par service). Nous avons montré qu'en utilisant des traces provenant de différents environnements pour la construction du modèle nous pouvions obtenir une classification plus robuste à ces changements d'environnement. Cependant, il faut noter que la gestion des différentes configurations est sûrement le problème le plus difficile à traiter au vu de la grande diversité des configurations possibles.

En conclusion, H2Classifer est une solution robuste qui s'adapte à différents services et peut être utilisé dans le temps. Il est cependant nécessaire d'être vigilant lors de sa mise en place, afin de privilégier la supervision de configurations majeures pour garantir son fonctionnement.

Conclusion générale

Le déploiement généralisé des communications chiffrées fait apparaître un besoin antagoniste d'outils de supervision capables de fonctionner malgré la présence du chiffrement, tels que la détection d'attaques ou de mauvais comportements dans des canaux chiffrés. Le protocole HTTPS qui est utilisé dans les navigateurs en est une parfaite illustration. Par son adoption et par son impact sur le grand public, de nombreux travaux de recherche se sont intéressés à la supervision de ce protocole selon différents niveaux de détection : reconnaissance des protocoles, des services ou des usages au sein d'un service. De notre côté, nous nous sommes principalement focalisé sur la détection des usages. Nous avons ainsi mis au point des solutions permettant de superviser certaines activités utilisateur dans du trafic HTTPS et respectant les trois contraintes que nous nous étions fixées :

— Passive : l'analyse s'effectue via l'écoute du trafic qui transite sur le réseau.

— Transparente : aucune installation n'est faite sur les terminaux ni les applications des utilisateurs.

— Respect de la vie privée : une telle solution ne doit rien divulguer sur une utilisation légitime d'un service.

Travail réalisé

Nous avons décrit dans le manuscrit de thèse, nos deux principales contributions. Chacune de ces contributions définit une nouvelle solution de supervision pour détecter la recherche de mots-clés non légitimes pour un service sécurisé avec HTTPS. La première étudie le trafic HTTPS lors de son utilisation avec la version 1.1 du protocole HTTP. La deuxième contribution traite, quant à elle, de la supervision du trafic HTTPS lorsqu'il est couplé avec le protocole HTTP/2.

Ces deux contributions ont été construites en trois étapes :

1. Analyse et modélisation du trafic
2. Extraction de caractéristiques et construction d'une solution à base de reconnaissance de signatures
3. Évaluation de la solution avec génération d'un jeu de données

Nous résumons ci-dessous ces trois étapes pour chacune d'elles.

Analyse de trafic chiffré HTTP/1.1

Via l'analyse du fonctionnement du protocole HTTP/1.1 lorsqu'il est couplé avec TLS, nous avons pu remarquer qu'il était possible de reconstruire la taille des objets HTTP chiffrés transitant sur le réseau. En utilisant également la taille des requêtes associées aux objets HTTP chiffrés, nous avons montré que nous pouvions discerner des objets spécifiques, comme les images

envoyées par un moteur de recherche d'images (en l'occurrence Google Images). Partant de cette observation, nous avons démontré théoriquement qu'il était possible de caractériser le trafic issu du chargement de la page associée à un mot-clé via la séquence des tailles des images chiffrées la composant. Nous avons construit alors une signature relative à un mot-clé donné en utilisant plusieurs traces issues du chargement de ce même mot-clé. Cela permet d'agréger les variations constatées entre ces différentes traces. La construction de cette signature utilise l'estimation par noyau (KDE). Afin de reconnaître l'usage d'un mot-clé non légitime dans une trace, nous avons mis en place une métrique basée sur le calcul d'un score pour l'ensemble des signatures. Cette métrique permet de désigner le mot-clé le plus probable. Ensuite la méthode filtre, grâce à une autre métrique, si la trace est légitime. Sinon, le mot-clé précédemment reconnu est retourné par la solution.

Nous avons développé une implémentation de cette solution que nous avons nommée H1Classifier afin d'évaluer les performances de classification de la méthode. Un large jeu de données de traces réseau a été collecté pour cette occasion sur le service Google Images. Ces traces nous ont permis d'étudier, lors d'une première expérience, l'impact des paramètres de la méthode de classification. Dans un deuxième temps, nous avons évalué le passage à l'échelle de H1Classifier. Dans le cadre de cette expérience, nous avons obtenu un taux de justesse de 99,18% (TPR = 0,966 et FPR = 0,0056) lors de la détection de 10 500 mots-clés non légitimes parmi 105 000 traces de mots-clés légitimes. Enfin, nous avons étudié l'adaptation de H1Classifier à du trafic HTTP/2. Pour cela, nous avons dû adapter les caractéristiques du trafic utilisé car le protocole HTTP/2 ne nous permet plus de reconstruire la taille des objets HTTP chiffrés. Cependant, les résultats obtenus n'étaient pas satisfaisants et nous avons conclu que cette méthode n'était pas adaptée à ce type de trafic. L'introduction du multiplexing dans HTTP/2 a modifié le trafic généré en profondeur et c'est pourquoi nous avons proposé une deuxième solution pour la supervision de ce trafic.

Analyse de trafic chiffré HTTP/2

Afin de définir une nouvelle méthode pour le protocole HTTP/2, nous avons défini un ensemble de caractéristiques capables de distinguer les traces générées par des mots-clés différents. Pour ce faire, nous avons utilisé un ensemble de caractéristiques issues de l'état de l'art, ainsi que de nouvelles caractéristiques provenant de nos observations du protocole HTTP/2 (fréquence des tailles de records, volume et variations du trafic). Nous avons construit un modèle permettant la reconnaissance de chaque mot-clé à superviser, utilisant de l'apprentissage supervisé et, en particulier, l'algorithme des forêts d'arbres décisionnels (RF). Notre méthode de classification combine ensuite, notre ensemble de caractéristiques associé à l'algorithme RF.

Nous avons réalisé plusieurs évaluations de H2CLassifier (implémentation de notre méthode de classification) permettant de mesurer ses performances dans différentes situations. Nous avons, tout d'abord, étudié l'impact des différents paramètres sur les capacités de classification pour déterminer un ensemble de paramètres optimaux (méta-configuration). Dans un deuxième temps, nous avons mesuré les variations de la performance H2Classifier en fonction du nombre de mots-clés à superviser ainsi que du nombre de traces disponibles lors de l'apprentissage pour tester le passage à l'échelle de la solution. Cette évaluation a été effectuée sur quatre services très utilisés. Dans le cas de la supervision de 2000 mots-clés non légitimes parmi un large ensemble de traces légitimes, H2Classifier a permis la classification de ces traces avec un TPR compris entre 61% et 98% (en fonction du service). Cependant, il est important de noter que cette classification discerne parfaitement les traces issues de mots-clés légitimes par rapport à celles qui ne le sont pas. En effet, nous observons pour cela un TPR de 99,9% avec un FPR inférieur à 0,1 ce qui

montre les bonnes performances de la solution. Cependant, certaines limitations de la solution, telles que sa robustesse, sa facilité de mise en œuvre et sa généricité, restent à évaluer.

La deuxième évaluation que nous avons conduite a mesuré l'impact du temps sur les performances de la classification. Pour cela, nous avons quotidiennement collecté des traces sur une durée de quatre mois pour quatre services. Avec ces données, nous avons constaté une décroissance progressive (prévisible) des résultats au cours du temps. Pour contrer cette dégradation, nous avons mis en place un réapprentissage régulier afin de stabiliser les performances dans le temps. Grâce à un réapprentissage tout les cinq jours, nous avons pu constater une stabilisation des résultats dans le temps. Le réapprentissage n'étant pas coûteux, cela permet à notre solution de rester valable dans le temps. Ainsi, moyennant une collecte de traces régulières, il n'est pas coûteux de maintenir à jour la solution de supervision, ce qui atteste donc de sa robustesse dans le temps.

La troisième évaluation est dédiée à la portabilité de H2Classifier pour la supervision de nouveaux services. A nouveau, nous avons construit un ensemble avec plus de traces réseau, mais, cette fois-ci, pour environ 3000 nouveaux services protégés par HTTPS (HTTP/2). Sur plus de 50% des nouveaux services évalués, H2Classifier permet une classification avec un taux de justesse supérieur ou égal à 90%. A l'inverse, pour seulement 10% des services, ce taux de justesse est inférieur à 50%. Ces résultats montrent ainsi la bonne capacité de H2Classifier à s'adapter à de nouveaux services. De plus, l'ensemble de ces résultats a été obtenu en utilisant la même méta-configuration définie précédemment, ce qui montre la facilité de mise en œuvre de la solution.

Finalement, nous avons évalué l'impact de l'environnement de capture sur les performances de la classification. Pour cela, nous avons collecté un ensemble de traces couvrant différentes configurations pour le navigateur ainsi que le service Google Images. Les résultats montrent que l'environnement de capture peut avoir un impact non négligeable sur les résultats, mais que ces derniers sont difficilement prévisibles a priori (spécifiques au service). Nous avons cependant observé que l'usage de traces issues de différents environnements lors de l'apprentissage du modèle permet l'obtention d'une classification plus robuste à ces changements.

Perspectives

Cette thèse ouvre de nombreuses perspectives dans le domaine de l'analyse de trafic chiffré en général.

Les contre-mesures à l'analyse de trafic

Dans ce manuscrit, nous avons adopté le point de vue du défenseur où les administrateurs souhaitent superviser leur trafic pour rester en conformité avec la loi. Cependant, l'analyse de trafic chiffré peut également être utilisée par un attaquant pour détecter l'utilisation spécifique d'un ou plusieurs services. En effet, la définition de la liste des mots-clés est libre et peut donc s'apparenter à de la surveillance ciblée de certains usages. L'analyse de trafic n'attaque pas directement le chiffrement mais observe la « forme » du trafic produite par l'application sous-jacente au chiffrement. Cependant, pour masquer cette forme du trafic, il est nécessaire d'utiliser une solution intermédiaire avant la phase de chiffrement afin d'ajouter du « bruit ». Il est intéressant de noter que HTTP/2 intègre déjà des options pour ajouter du padding mais cette option n'est pas utilisé dans le trafic courant.

Dans l'état de l'art, des travaux traitent cette problématique et propose des solutions telles que [76, 122, 75, 67, 78]. Ces méthodes ont cependant des résultats variables en fonction du

protocole étudié et présentent un surcoût en bande passante comprise entre 10% et 190%. Il serait ainsi très intéressant d'observer, dans un premier temps, comment nos solutions réagissent à la présence de telles contre-mesures. Ensuite, dans un deuxième temps, il conviendrait de trouver un équilibre optimal en termes de ressources utilisées et difficultés pour analyser le trafic chiffré. La légèreté d'une telle solution est la contrainte majeure pour son adoption car la tendance actuelle vise à privilégier la performance des échanges.

Domaines d'application

Dans cette thèse, nous avons défini notre étude autour du protocole HTTPS. Cependant, les méthodes d'analyse de trafic peuvent s'appliquer à toutes sortes d'applications ou de protocoles comme nous avons déjà pu le voir dans l'état de l'art. Durant cette thèse, nous avons débuté l'investigation de la détection d'actions utilisateur au sein d'une application de messagerie instantanée. L'objectif était de détecter certaines actions telles que l'envoi ou la réception d'un message, l'envoi ou la réception d'une pièce jointe, l'envoi ou la réception d'un appel audio/vidéo, etc. Cependant, ces travaux nécessitent de relever plusieurs défis importants, tels que la génération et la capture automatisées des comportements utilisateurs, la compréhension du protocole de l'application, souvent propriétaire, et la construction de signatures fiables pour des comportements générant peu de trafic. Il serait donc possible de suivre le même cheminement méthodologique que pour HTTPS, mais ce cheminement est difficilement automatisable pour les raisons évoquées précédemment.

Défis futurs

Les solutions que nous avons présentées dans cette thèse n'avaient pas pour vocation première la performance. Un des challenges futurs est ainsi de les optimiser pour réaliser de la supervision en temps réel. De plus, l'intégration de ces solutions dans un système tel qu'un IDS impose également de revoir les processus de sécurité standards. D'une part, ces nouvelles solutions apportent de l'information incomplète par rapport à des approches travaillant avec du trafic en clair. D'autre part, l'utilisation d'intelligence artificielle apporte des résultats soumis à des probabilités dont il faut tenir compte pour lever ou non des alertes.

Un autre challenge futur est le support des solutions telles que : QUIC, HTTP/3 et TLS 1.3. Ces technologies sont actuellement en cours de standardisation mais sont déjà utilisées par certains services et supportées par les navigateurs. Il sera ainsi intéressant d'étudier comment les solutions que nous avons proposées pourront s'adapter à ces nouvelles technologies.

Productions

Publications

Les contributions présentées dans ce mémoire ont donné lieu à plusieurs publications scientifiques :

— **Transparent and Service-Agnostic Monitoring of Encrypted Web Traffic**,
publié dans IEEE TNSM 16(3) [123], décrit la méthodologie ainsi que les résultats obtenues pour la classification du trafic HTTP/1.1, présentées dans le chapitre 3 et le début du chapitre 4.

— **Passive Monitoring of HTTPS Service Use**,
publié à CNSM 2018 [124], correspond à la présentation des deux méthodes de classifications (HTTP/1.1 et HTTP/2) ainsi qu'à l'évaluation de ces dernières. Ces travaux correspondent aux chapitres 3 à 5 ainsi que le début du chapitre 6.

— **Encrypted HTTP/2 Traffic Monitoring : Standing the Test of Time and Space**,
publié à WIFS 2020 [125], présente en détails les évaluations de H2Classifier sur le temps et sur les nouveaux services. Ces travaux correspondent aux sections 6.3 et 6.4 du chapitre 6.

— **Analyse de Trafic Chiffré : Application au Web**,
publié à RESSI 2018 [126], présentation ses travaux relatifs à HTTP/1.1 dans une conférence francophone.

Prototypes et jeux de données

Prototypes :

— **H1Classifier** : Implémentation en python de la méthode de classification du trafic HTTP/1.1 chiffré décrite dans le chapitre 3. Elle utilise la librairie scikit-learn [111] pour la méthode KDE.

— **H2Classifier** : Implémentation en python de la méthode de classification du trafic HTTP/2 chiffré décrite dans le chapitre 5. Elle utilise la librairie scikit-learn [111] pour l'algorithme des forêts d'arbres aléatoires.

— **Solution de capture** : Implémentation en python de la capture des traces réseaux correspondant au chargement des pages web issue de la recherche d'un mot-clé. Elle utilise la librairie Selenium pour l'automatisation du comportement utilisateur dans le navigateur et TCPdump pour la capture des traces réseau.

Jeux de données :

— $data_1$ **et** $data_2$ **(HTTP/1.1)** : 157 500 traces réseau correspondant à la recherche d'un mot-clé sur le moteur de recherche Google Images avec HTTP/1.1. Il se compose de 5

traces pour 10 500 mots-clés ainsi que une trace pour 105 000 autres mots-clés. Lien vers la page du jeu de données : `http://betternet.lhs.loria.fr/datasets/googleimg/`.

— **data_h2_500 et data_h2_2000 (HTTP/2)** : Deux ensembles de traces réseaux issue de la capture du flux chiffré HTTP/2 généré par la recherche de différents mots clés sur plusieurs services web (Amazon, Google, Google Images et Google Maps). data_h2_500 dispose de 60 captures pour chacun des 500 mots-clés par service. data_h2_2000 dispose lui de 12 traces pour chacun des 2000 mots-clés pour chaque service. Il contient également 20 000 traces correspondant à d'autres mots-clés pour chacun des services. Lien vers la page du jeu de données : `http://betternet.lhs.loria.fr/datasets/h2classifier/`.

— **data_temporelle** : 968 000 traces réseau correspondant à la capture quotidienne de 4 traces pour la recherche de 500 mots-clés sur 4 services pendant 121 jours. Lien vers la page du jeu de données : `http://betternet.lhs.loria.fr/datasets/h2classifier/`.

— **data_portabilité** : 1 238 400 traces réseau correspondant à la capture de 20 traces issue de la recherche de 20 mots-clés sur plus de 3000 services différents. Lien vers la page du jeu de données : `http://betternet.lhs.loria.fr/datasets/h2classifier/`.

Glossaire

AES Advanced Encryption Standard : algorithme de chiffrement symétrique

ALPN Application-Layer Protocol Negotiation : une extension du protocole TLS (RFC 7301)

ANSSI Agence Nationale de la Sécurité des Systèmes d'Information

ASCII American Standard Code for Information Interchange ou code américain normalisé pour l'échange d'information

Acc Accuracy ou Justesse

CART ClAssification and Regression Trees : une méthode de construction d'arbres décisionnels

CERN Conseil Européen pour la Recherche Nucléaire

CRLF Retour chariot (CR) et saut de ligne (LF)

DNS Domain Name System : service distribué qui traduit un nom de domaine en adresse IP

DPI Deep packet inspection ou analyse de paquet profond

DT Decision Tree ou Arbres décisionnels

DoH DNS over HTTPS ou DNS sur HTTPS

DoT DNS over TLS ou DNS sur TLS

EM Espérance-Maximisation

FIFO First In, First Out ou premier entré, premier sorti

FNR False Negative Rate ou Taux de faux négatifs

FPR False Positive Rate ou Taux de faux positifs

GMM Gaussian Mixture Model : une méthode de calcul de densité

HAR HTTP ARchive : un format ouvert destiné à l'export et l'échange de données collectées par des outils de monitoring HTTP

HMAC Hash-based Message Authentication Code ou code d'authentification d'une empreinte cryptographique de message avec clé

HPACK Protocole de compression des entêtes pour HTTP/2 (RFC 7541)

HTML HyperText Markup Language : langage conçut pour représenter le contenu et la structure d'une page web

HTTPS HyperText Transfer Protocol Secure ou Protocole de transfert hypertextuel sécurisé : combinaison des protocoles HTTP et TLS

HTTP HyperText Transfer Protocol ou Protocole de transfert hypertextuel

IANA Internet Assigned Numbers Authority : autorité officiel supervise l'allocation des adresses IP, des numéros de ports TCP ou les racines DNS

IDS Intrusion Detection System ou système de détection d'intrusion

IETF Internet Engineering Task Force : organisme qui élabore les standards Internet

IMAPS Internet Message Access Protocol : protocole pour le courrier électronique qui permet l'accès direct à ces derniers sur le serveur de messagerie

IP Internet Protocole : protocole internet

JAP Java Anon Proxy : un proxy d'anonymisation

KDE Kernel Density Estimation : une méthode de calcul de densité

kNN k-nearest neighbors ou méthode des k plus proches voisins

LDAP Lightweight Directory Access Protocol : protocole pour la gestion des données des utilisateurs d'un réseau

MTU Maximum Transmission Unit : taille maximale d'un paquet TCP sans fragmentation

NB classification Naïve Bayésienne

NN Neural Network ou réseax de neurones

OS Operating System ou système d'exploitation

PA Proxy d'Anonymisation

QUIC Quick UDP Internet Connections : protocole de transport rapide basé sur UDP

RDC Réseau de distribution de contenus

RFC Request For Comments : documents de référence qui décrit et spécifie une norme ou un standard

RF Random Forest ou Forêt d'arbres aléatoires

ROC Receiver Operation Characteristic ou fonction d'efficacité du récepteur

SMTPS Simple Mail Transfer Protocol Secure : protocole pour échanger le courrier électronique

SNI Server Name Indication : entête TLS qui indique le nom de l'hôte que l'on souhaite contacter

SPDY protocole réseau applicatif qui a inspiré fortement le protocole HTTP/2

SSH Secure SHell :protocole réseau sécurisé qui permet l'administration à distance d'une machine

SSL Secure Sockets Layer ou Couche de sockets sécurisée : protocole prédécesseur de TLS

SVM Support Vector Machine ou machines à vecteurs de support

TCP Transmission Control Protocol ou protocole de contrôle de transmissions : protocole de transport fiable

TLS Transport Layer Security : protocole de sécurité pour le chiffrement des échanges sur Internet

TNR True Negative Rate ou Taux de vraies négatifs

TOR The Onion Router : un proxy d'anonymisation

TPR True Positive Rate ou Taux de vraies positifs

UDP User Datagram Protocol ou protocole de datagramme utilisateur : un des principaux protocoles de communication utilisés sur Internet

URL Uniform Resource Locator : une adresse (web) permettant la localisation de ressources sur Internet

VBR Variable bitrate ou taux d'échantillonnage variable

VPN Virtual Private Network ou réseau privé virtuel

WCR Wrong Classification Rate ou Taux de mauvaises classifications

WF Website Fingerprinting ou Détection de page web par reconnaissance de signature

Table des figures

1 Pourcentage des pages HTTPS ouvertes avec un navigateur en fonction du temps 2
2 Coût annuel moyen par type de conséquences d'une attaque cyber (extrait de [2]) 3
3 Évolution de l'adoption de HTTP/2 (données provenant du w3techs) 5
4 Schématisation de la problématique 6

1.1 Modèle TCP/IP et positionnement des protocoles 12
1.2 Évolution de SSL/TLS dans le temps (Extrait du rapport Enisa 2018 sur le chiffrement [7]) 14
1.3 Handshake TLS avec authentification du serveur 15
1.4 Entête TLS (en bits) 15
1.5 TLS record layer 16
1.6 Couches protocolaires en fonction des versions de HTTP (avec chiffrement) . . . 19
1.7 Établissement d'une connexion HTTP/1.1 ou HTTP/2 20
1.8 Récapitulatif des différents modes d'échanges de HTTP 22
1.9 Structure générique d'un message pour HTTP/1.1 23
1.10 Structure HTTP/2 (tailles en bits) 25
1.11 Exemple de frames (tailles en bits) 26

2.1 Découpage des différents niveaux de classification 30
2.2 Schématisation de la solution Blindbox (d'après J. Sherry [90]) 37

3.1 Courbe cumulative du nombre d'images en fonction de la taille 47
3.2 Architecture globale de la solution (HTTP/1.1) 49
3.3 Schématisation des étapes de la phase d'extraction des caractéristiques 50
3.4 Exemples de distribution de la tailles des requêtes pour la recherche de deux mots-clés différents sur le service Google Images 51
3.5 Exemple d'application de GMM avec 3 composantes 53
3.6 Exemple d'application de KDE avec différents paramètres h 54
3.7 Comparaison du temps de calcul (apprentissage et test) pour les algorithmes KDE et GMM 56
3.8 Comparaison du temps de calcul (apprentissage et test) pour les algorithmes KDE et GMM avec un nombre de tailles par signature élevé 57
3.9 Positionnement et composition de l'étape de classification dans la phase de supervision 58

4.1 Schéma outil de capture 63
4.2 Définition des métriques 65
4.3 Courbe ROC : évaluation de la largeur de fenêtre h ($data_1$) 67

4.4 Impact du paramètre fenêtre (h) sur le TPR et le FPR en faisant varier le paramètre α ($data_1$) . . . 68
4.5 Impact du nombre de mots-clés non légitimes considérés sur les résultats de la classification ($data_1$) . . . 69
4.6 Courbe ROC pour le test à grande échelle ($data_1$ et $data_2$) . . . 70
4.7 Protocole record TLS . . . 74
4.8 Exemple de fragmentation avec un ou deux objets dans le tampon d'entrée du protocole TLS . . . 76
4.9 Calcul de dissimilarité pour des fonctions signatures avec du trafic HTTP/1.1(1000 mots-clés) sur le service Google Images . . . 77
4.10 Calcul de dissimilarité pour des fonctions signatures avec du trafic HTTP/2 . . . 78

5.1 Caractéristiques pour l'identification de traces HTTP/2 . . . 85
5.2 Architecture globale de la solution (trafic HTTP/2) . . . 88
5.3 Schématisation du processus d'extraction des caractéristiques pour le trafic HTTP/2 89
5.4 Exemple d'utilisation de CART pour un problème à deux variables c^1 et c^2 (d'après R. Genuer [119]) . . . 93
5.5 Schéma globale de l'algorithme RF . . . 94

6.1 TPR pour l'évaluation des paramètres $n_estimators$ et max_depth . . . 103
6.2 Évaluation du paramètre n_feats . . . 105
6.3 FPR relative aux variations du paramètre n_train_leg . . . 106
6.4 Résultats de l'évaluation du paramètre $max_samples$ . . . 107
6.5 Récapitulatif des variations du TPR et WCR en fonction des expériences . . . 112
6.6 Justesse quotidienne moyenne par service . . . 115
6.7 Justesse moyenne chaque jour avec apprentissage tous les n_j jours. . . . 116
6.8 Justesse quotidienne moyenne avec apprentissage tout les 5 jours (en haut : les meilleurs 10% des mots-clés, en bas : les moins bons 10% des mots-clés) . . . 117
6.9 Catégories de 3096 services retenues . . . 120
6.10 Courbe cumulative de la justesse pour les 3096 services testés . . . 121

11 Amazon : max_depth et n_estimators . . . 142
12 Google : max_depth et n_estimators . . . 143
13 Google Images : max_depth et n_estimators . . . 144
14 Google Maps : max_depth et n_estimators . . . 145
15 Tout les services : n_feats . . . 146
16 Tout les services : n_train_leg . . . 147
17 Tout les services : max_samples . . . 148

Liste des tableaux

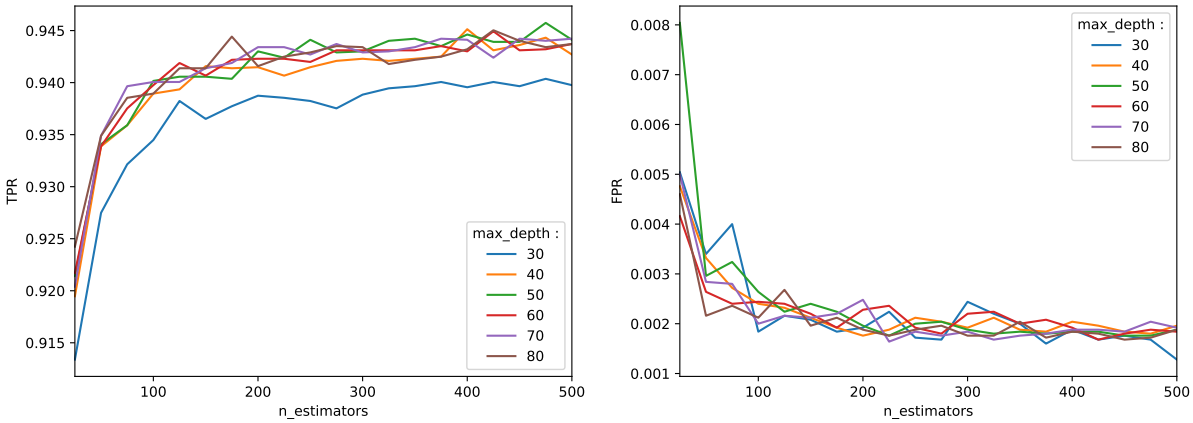
2.1 Références triées par niveau de détection 39

4.1 Jeu de données pour le trafic HTTP/1.1 64
4.2 Résultats $data_1$: 1050 mots-clés non légitimes / 8000 légitimes 70
4.3 Résultats $data_1$ & $data_2$: 10 500 mots-clés non légitimes / 105 000 légitimes . . . 71
4.4 Résultats pour H1Classifier avec du traffic HTTP/2 en monde fermé (moyenne 5 évaluations) 76

6.1 Jeu de données HTTP/2 : data_h2_2000 100
6.2 Jeu de données HTTP/2 : data_h2_500 101
6.3 Composition du jeu de données pour la phase de réglage des paramètres (m. = mots-clés et t. = traces) 102
6.4 Traces utilisées par service pour l'évaluation 1 108
6.5 Évaluation H2Classifier avec 500 mots-clés et 56 traces d'apprentissages par mot-clé (moyenne des 5 évaluations et écart-type entre parenthèses) 108
6.6 Composition par service pour l'expérience 2 109
6.7 Évaluation H2Classifer avec 500 mots-clés et 11 traces d'apprentissages par mot-clé (moyenne des 5 évaluations et écart-type entre parenthèses) 110
6.8 Composition par service pour l'expérience 3 (uniquement *data_h2_2000*) 110
6.9 Évaluation H2Classifier avec 2000 mots-clés et 11 traces d'apprentissages par mot-clé (moyenne des 5 évaluations et écart-type entre parenthèses) 111
6.10 Jeu de données temporel 114
6.11 Jeu de données des services 118
6.12 Résultats sur l'impact de l'environnement 122
6.13 Résultats sur l'impact de l'environnement en intégrant des traces de la nouvelle configuration durant l'entraînement (moyenne sur 5 expériences) 123

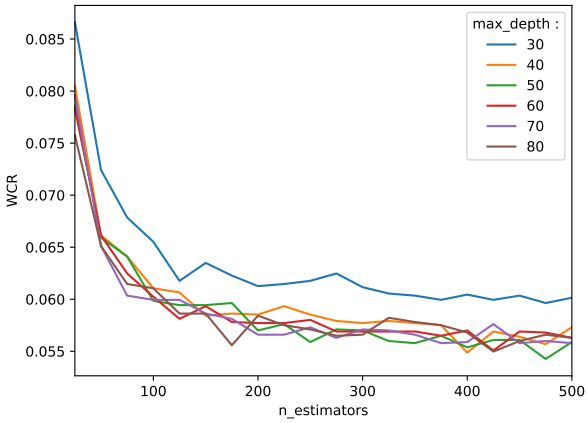
Annexe

Réglages des paramètres H2Classifer (figures)

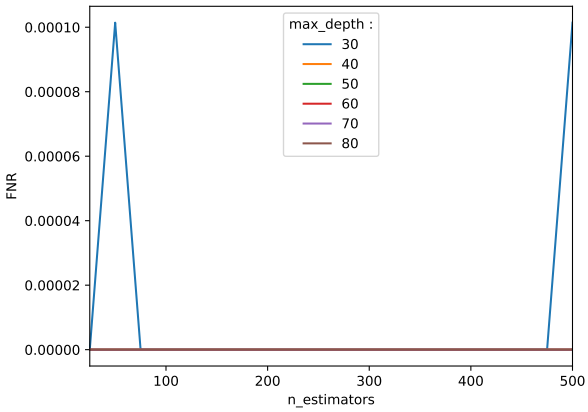


(a) TPR

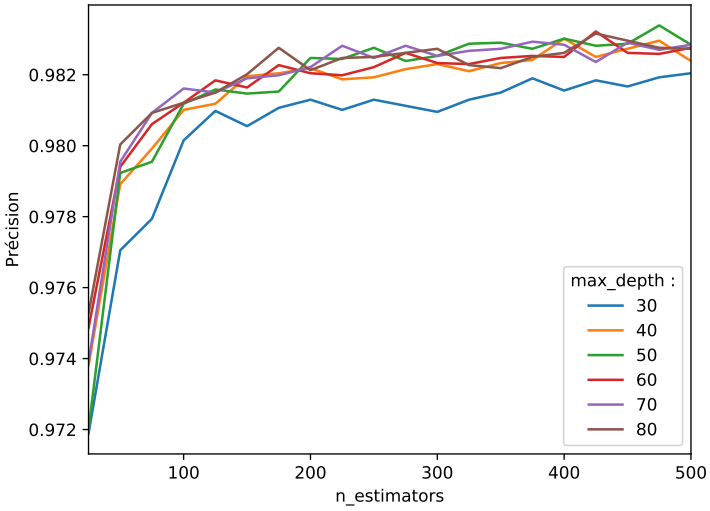
(b) FPR



(c) WCR

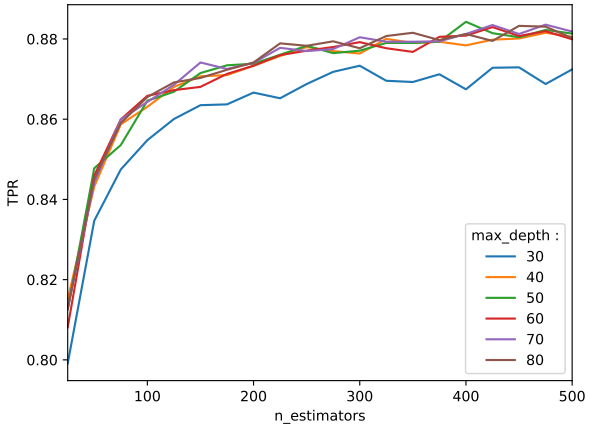


(d) FNR

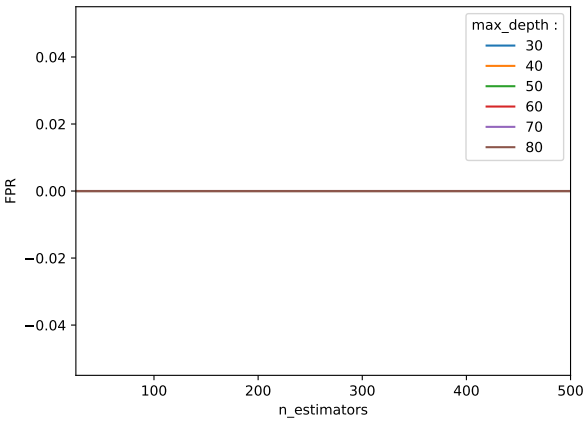


(e) Précision

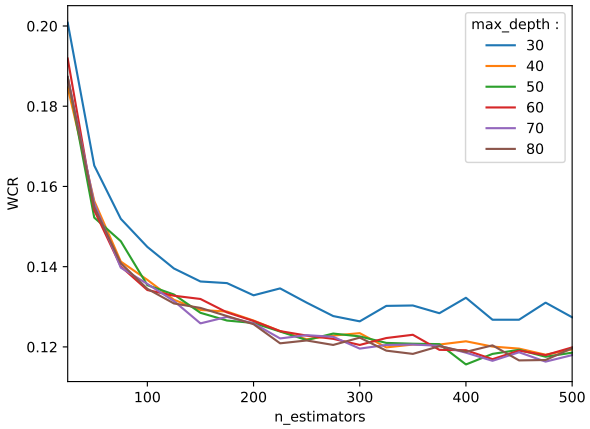
FIGURE 11 – Amazon : max_depth et n_estimators



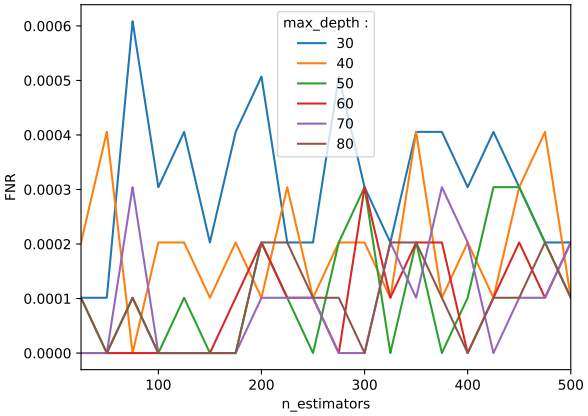
(a) TPR



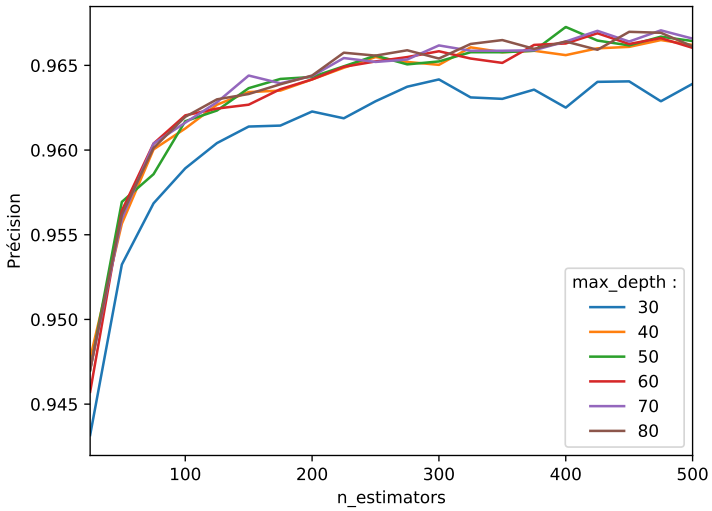
(b) FPR



(c) WCR

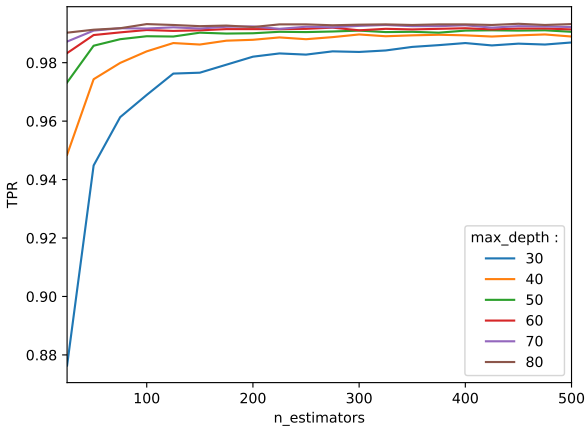


(d) FNR

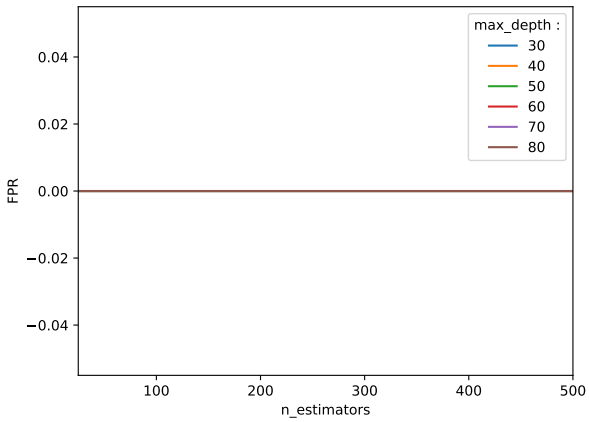


(e) Précision

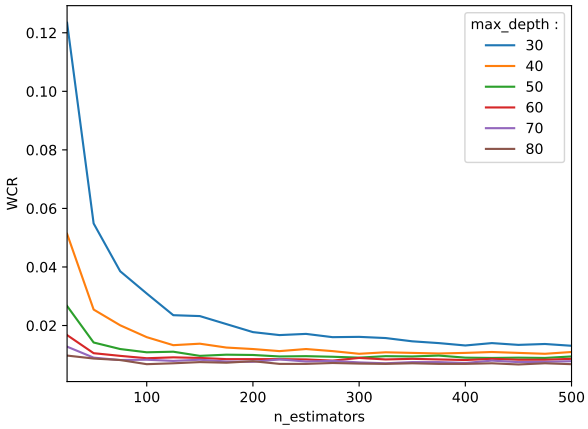
FIGURE 12 – Google : max_depth et n_estimators



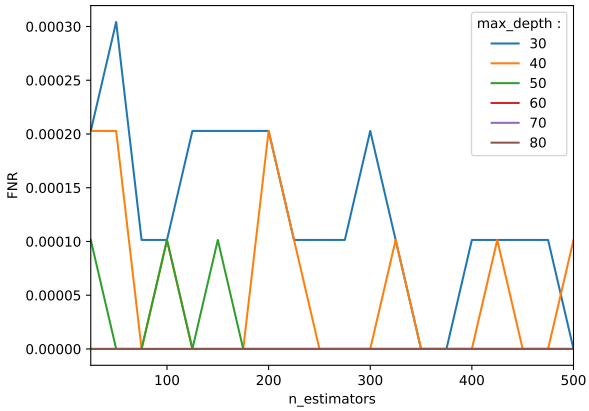
(a) TPR



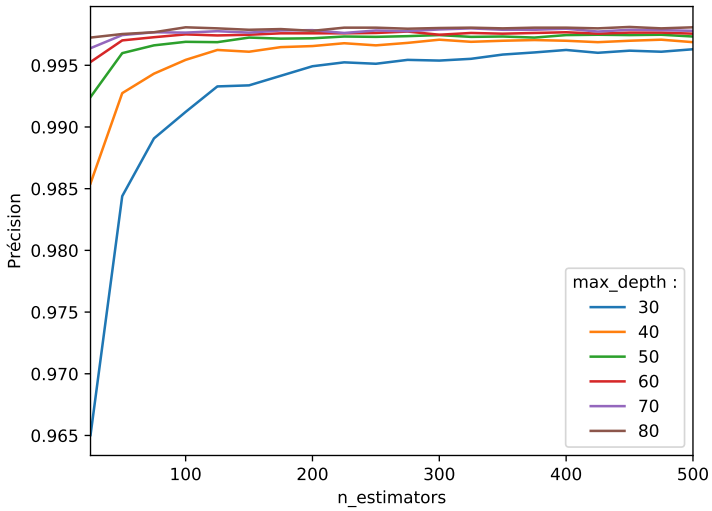
(b) FPR



(c) WCR

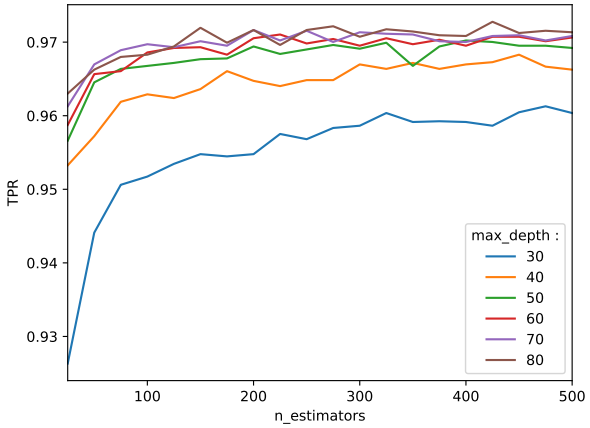


(d) FNR

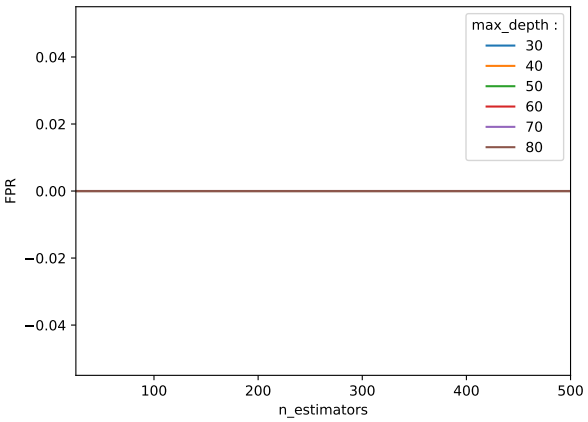


(e) Précision

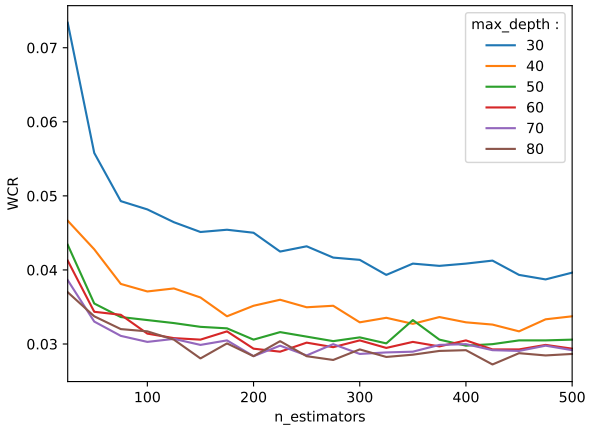
FIGURE 13 – Google Images : max_depth et n_estimators



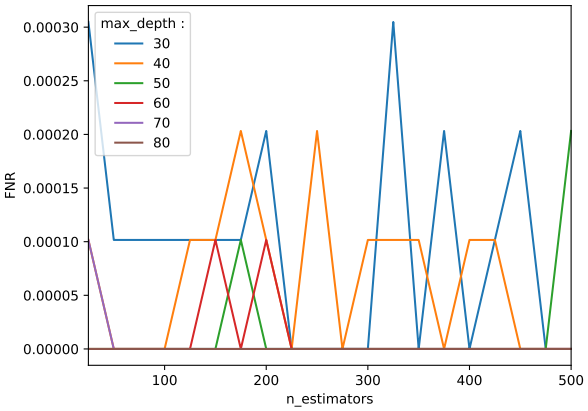
(a) TPR



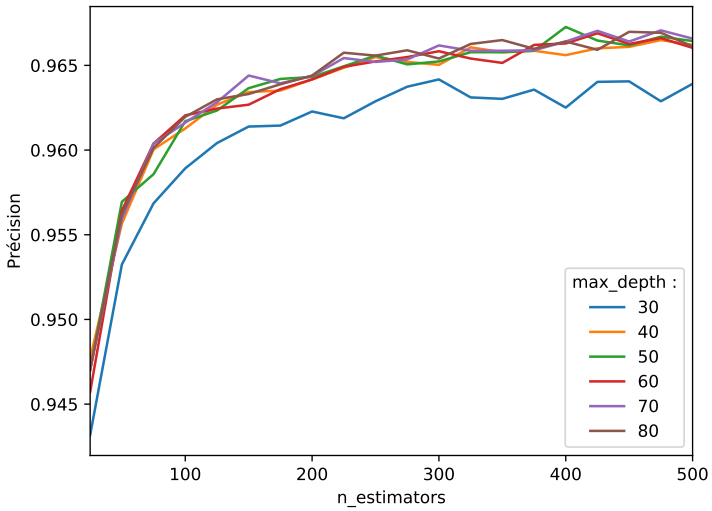
(b) FPR



(c) WCR

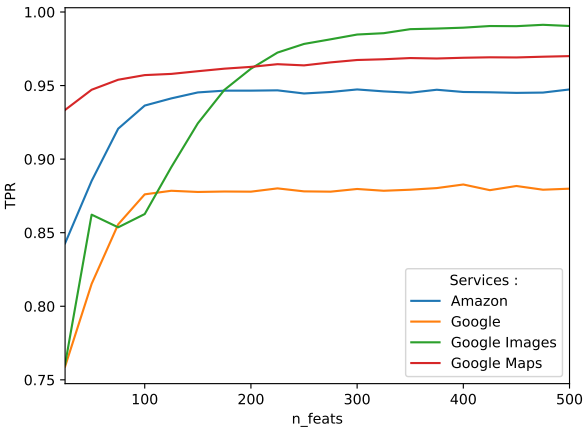


(d) FNR

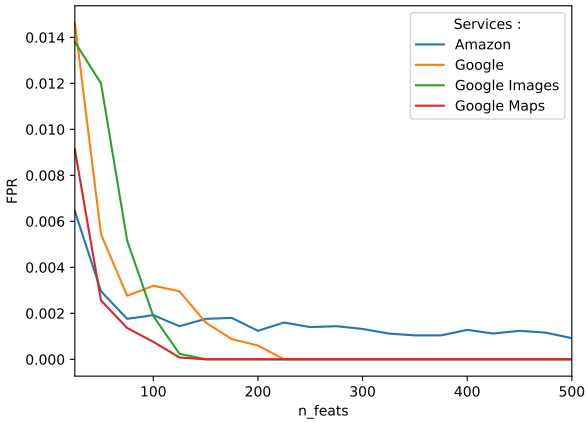


(e) Précision

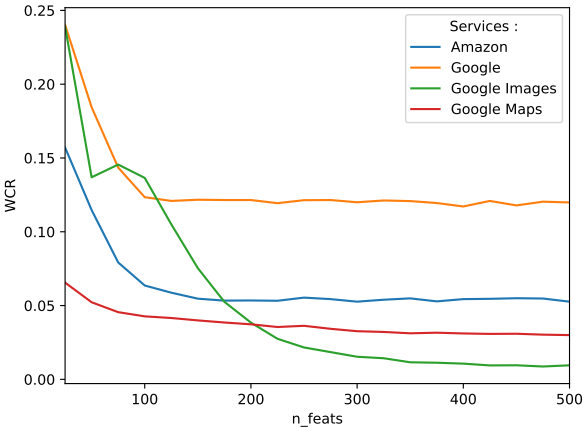
FIGURE 14 – Google Maps : max_depth et n_estimators



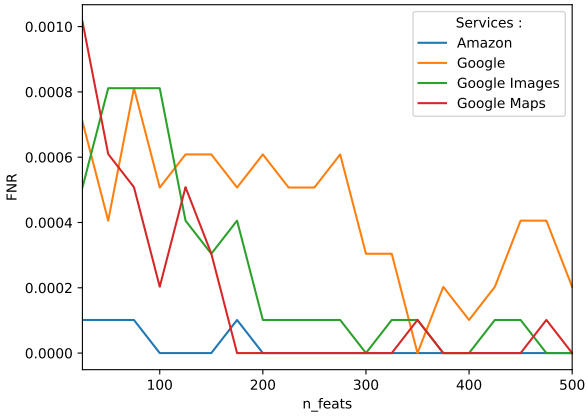
(a) TPR



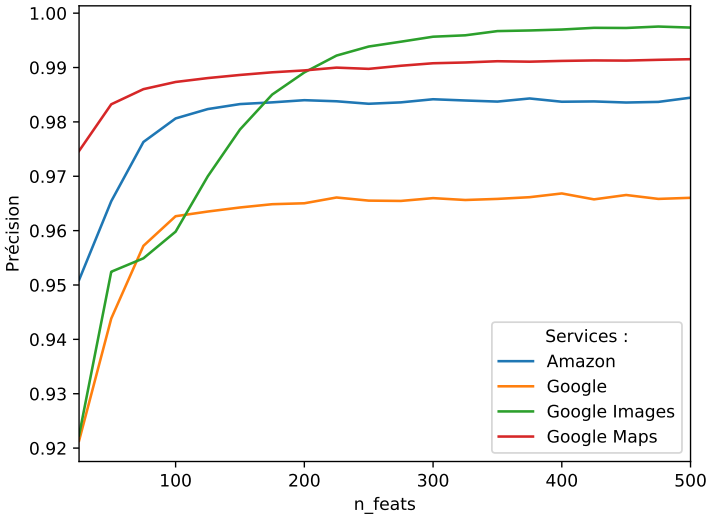
(b) FPR



(c) WCR

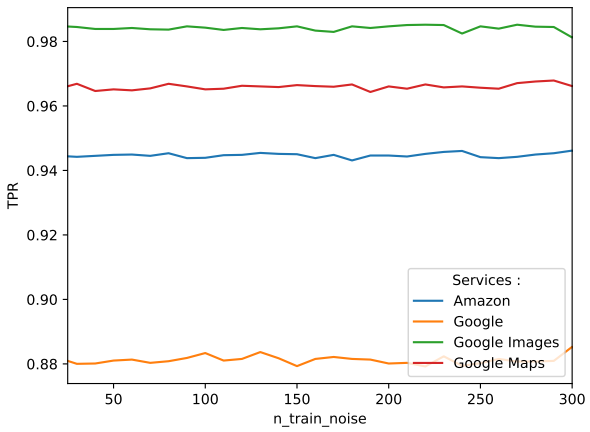


(d) FNR

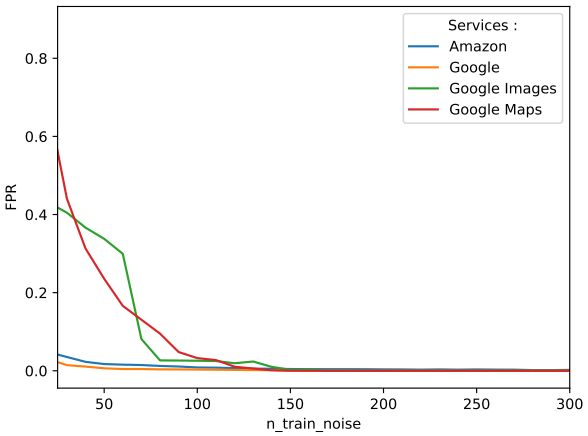


(e) Précision

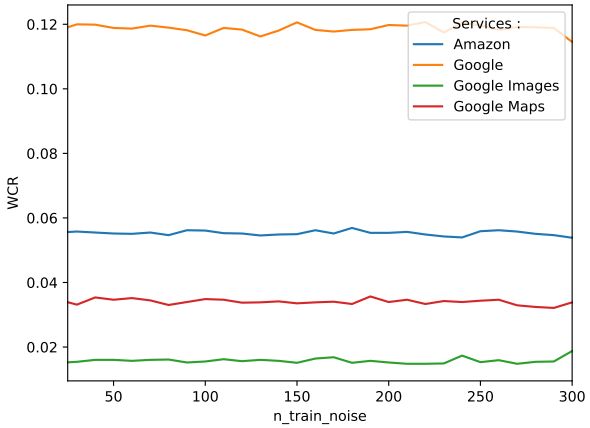
FIGURE 15 – Tout les services : n_feats



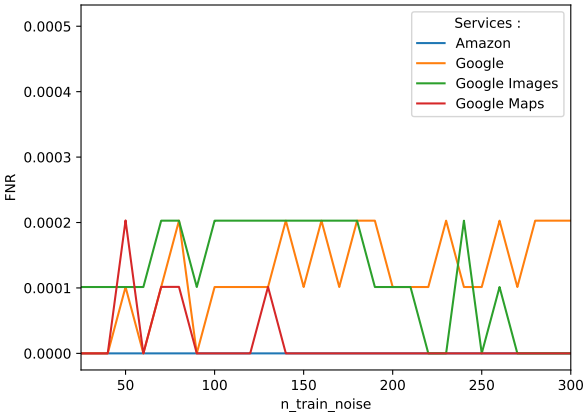
(a) TPR



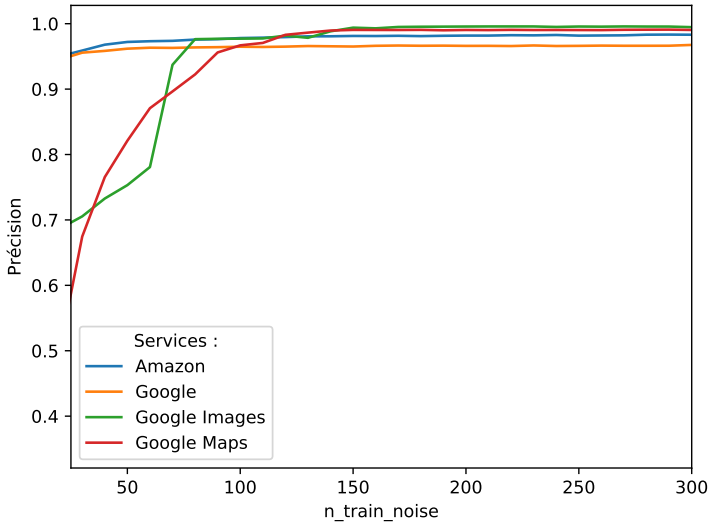
(b) FPR



(c) WCR

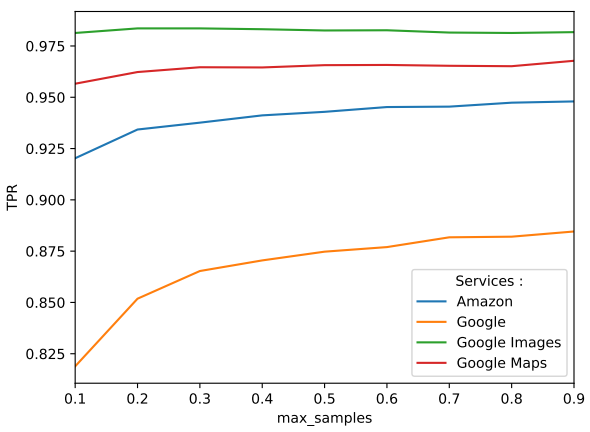


(d) FNR

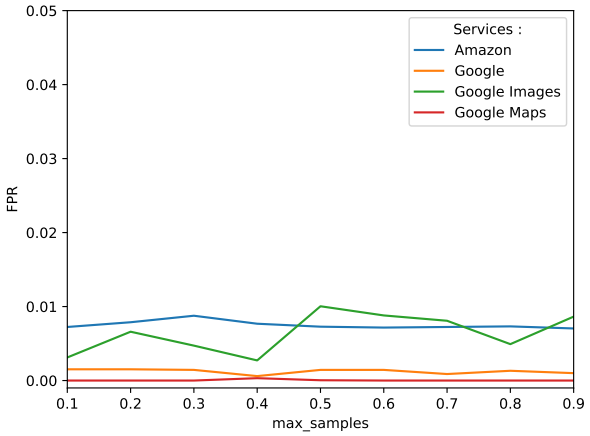


(e) Précision

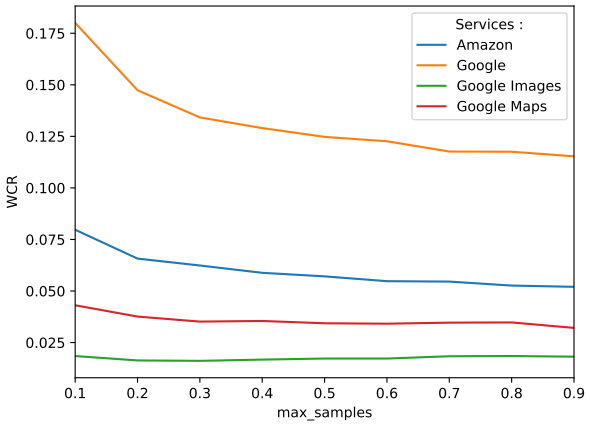
FIGURE 16 – Tout les services : n_train_leg



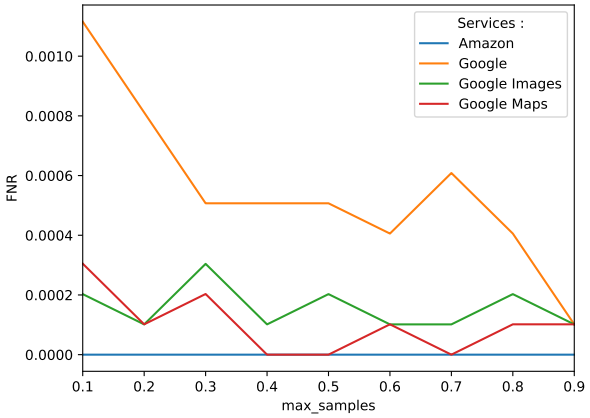
(a) TPR



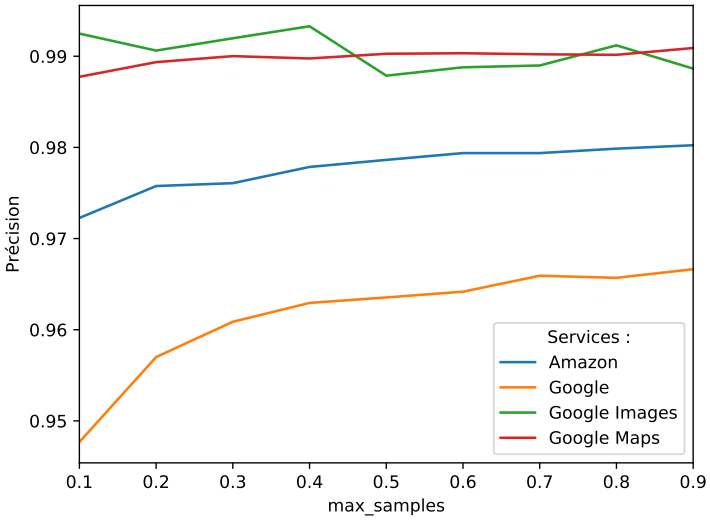
(b) FPR



(c) WCR



(d) FNR



(e) Précision

FIGURE 17 – Tout les services : max_samples

Bibliographie

[1] R. Peon and M. Thomson, "Hypertext transfer protocol version 2 (http/2)," RFC 7540, 2015.

[2] K. Bissell and L. Ponemon, "9th annual cost of cybercrime study," Accenture, Tech. Rep., 2019.

[3] E. Rescorla, "Http over tls," RFC 2818, 2000.

[4] S. Turner, "Prohibiting secure sockets layer (ssl) version 2.0," RFC 6176, 2011.

[5] R. Barnes, "Deprecating secure sockets layer version 3.0," RFC 7568, 2015.

[6] E. Rescorla, "The transport layer security (tls) protocol version 1.3," RFC 8446, 2018.

[7] D. Paraskevi, "Encrypted traffic analysis : Use cases & security challenges," Enisa, Tech. Rep., November 2018.

[8] J. Rizzo and T. Duong, "The crime attack," in *ekoparty security conference*, vol. 2012, 2012.

[9] R. F. et al., "Hypertext transfer protocol – http/1.1," RFC 2068, 1997.

[10] ——, "Hypertext transfer protocol – http/1.1," RFC 2616, 1999.

[11] ——, "Hypertext transfer protocol – http/1.1," RFC 7230-7237, 2014.

[12] M. Belshe, R. Peon, M. Thomson, and A. Melnikov, "Spdy protocol," *IETF draft, draft-ietf-httpbishttp2-00*, 2012.

[13] J. Iyengar and M. Thomson, "Quic : A udp-based multiplexed and secure transport (draft)," draft-RFC, 2020.

[14] M. Bishop, "Hypertext Transfer Protocol Version 3 (HTTP/3)," Internet Engineering Task Force, Internet-Draft draft-ietf-quic-http-29, Jun. 2020, work in Progress. [Online]. Available : https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-29

[15] S. Friedl, A. Popov, A. Langley, and S. Emile, "Transport layer security (tls) application-layer protocol negotiation extension," RFC 7301, 2014.

[16] P. Angelo, H. Neal, and G. Yoel, "Ssl, gone in 30 seconds : A breach beyond crime." Blackhat, 2013.

[17] R. Peon, "Hpack : Header compression for http/2," RFC 7541, 2015.

[18] S. Valenti, D. Rossi, A. Dainotti, A. Pescapè, A. Finamore, and M. Mellia, "Reviewing traffic classification," in *Data Traffic Monitoring and Analysis*. Springer, 2013, pp. 123–147.

[19] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Muller, and K. Haenssgen, "A survey of payload-based traffic classification approaches," *IEEE Communications Surveys & Tutorials*, vol. PP, pp. 1–22, 12 2013.

[20] A. Callado, C. Kamienski, G. Szabó, B. P. Gero, J. Kelner, S. Fernandes, and D. Sadok, "A survey on internet traffic identification," *IEEE communications surveys & tutorials*, vol. 11, no. 3, pp. 37–52, 2009.

[21] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *Commun. Surveys Tuts.*, vol. 10, no. 4, p. 56–76, Oct. 2008. [Online]. Available : https://doi.org/10.1109/SURV.2008.080406

[22] H. Alizadeh and A. Zúquete, "Traffic classification for managing applications' networking profiles," *Security and Communication Networks*, vol. 9, no. 14, pp. 2557–2575, 2016.

[23] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A survey of methods for encrypted traffic classification and analysis," *International Journal of Network Management*, 2015.

[24] W. M. Shbair, "Service-level monitoring of https traffic," Ph.D. dissertation, 2017, thèse de doctorat dirigée par Chrisment, Isabelle et Cholez, Thibault Informatique Université de Lorraine 2017. [Online]. Available : http://www.theses.fr/2017LORR0029

[25] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification : A systematic survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1988–2014, 2019.

[26] R. Alshammari, P. I. Lichodzijewski, M. Heywood, and A. N. Zincir-Heywood, "Classifying ssh encrypted traffic with minimum packet header features using genetic programming," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference : Late Breaking Papers*, 2009, pp. 2539–2546.

[27] R. Alshammari and A. N. Zincir-Heywood, "A flow based approach for ssh traffic detection," in *2007 IEEE international conference on systems, man and cybernetics*. IEEE, 2007, pp. 296–301.

[28] ——, "Machine learning based encrypted traffic classification : Identifying ssh and skype," in *2009 IEEE symposium on computational intelligence for security and defense applications*. IEEE, 2009, pp. 1–8.

[29] ——, "Can encrypted traffic be identified without port numbers, ip addresses and payload inspection ?" *Computer networks*, vol. 55, no. 6, pp. 1326–1350, 2011.

[30] ——, "An investigation on the identification of voip traffic : Case study on gtalk and skype," in *2010 International Conference on Network and Service Management*. IEEE, 2010, pp. 310–313.

[31] M. Korczyński and A. Duda, "Classifying service flows in the encrypted skype traffic," in *2012 IEEE International Conference on Communications (ICC)*. IEEE, 2012, pp. 1064–1068.

[32] D. Zhang, C. Zheng, H. Zhang, and H. Yu, "Identification and analysis of skype peer-to-peer traffic," in *2010 Fifth International Conference on Internet and Web Applications and Services*. IEEE, 2010, pp. 200–206.

[33] L. Bernaille and R. Teixeira, "Early recognition of encrypted applications," in *Proceedings of the 8th International Conference on Passive and Active Network Measurement*, ser. PAM'07. Berlin, Heidelberg : Springer-Verlag, 2007, p. 165–175.

[34] M. Dusi, F. Gringoli, and L. Salgarelli, "Quantifying the accuracy of the ground truth associated with internet traffic traces," *Comput. Networks*, vol. 55, pp. 1158–1167, 2011.

[35] S. Kim, Y. Goo, M. Kim, S. Choi, and M. Choi, "A method for service identification of ssl/tls encrypted traffic with the relation of session id and server ip," in *2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2015, pp. 487–490.

[36] Changxing Liu, Guanglu Sun, and Yibo Xue, "Drpsd : An novel method of identifying ssl/tls traffic," in *World Automation Congress 2012*, 2012, pp. 415–419.

[37] G. Sun, Y. Xue, Y. Dong, D. Wang, and C. Li, "An novel hybrid method for effectively classifying encrypted traffic," in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, 2010, pp. 1–5.

[38] Ipoque, *PACE 2.0 product (website)*. [Online]. Available : https://www.ipoque.com/

[39] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "ndpi : Open-source high-speed deep packet inspection," in *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2014, pp. 617–622.

[40] CISCO, *NBAR product (website)*. [Online]. Available : https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/network-based-application-recognition-nbar/prod_case_study09186a00800ad0ca.html

[41] S. Alcock and R. Nelson, "Libprotoident : traffic classification using lightweight packet inspection," *WAND Network Research Group, Tech. Rep*, 2012.

[42] T. Bujlow, V. Carela-Español, and P. Barlet-Ros, "Independent comparison of popular dpi tools for traffic classification," *Computer Networks*, vol. 76, pp. 75–89, 2015.

[43] C. McCarthy and A. N. Zincir-Heywood, "An investigation on identifying ssl traffic," in *2011 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*. IEEE, 2011, pp. 115–122.

[44] W. M. Shbair, T. Cholez, J. Francois, and I. Chrisment, "A multi-level framework to identify https services," in *Network Operations and Management Symposium (NOMS)*. IEEE, 2016.

[45] C. V. Wright, F. Monrose, and G. M. Masson, "On inferring application protocol behaviors in encrypted network traffic," *Journal of Machine Learning Research*, vol. 7, no. Dec, pp. 2745–2769, 2006.

[46] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the https certificate ecosystem," in *Proceedings of the 2013 conference on Internet measurement conference*, 2013, pp. 291–304.

[47] P. Velan, J. Medková, T. Jirsík, and P. Čeleda, "Network traffic characterisation using flow-based statistics," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2016, pp. 907–912.

[48] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "Acas : automated construction of application signatures," in *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, 2005, pp. 197–202.

[49] M. Dudik, S. J. Phillips, and R. E. Schapire, "Performance guarantees for regularized maximum entropy density estimation," in *International Conference on Computational Learning Theory*. Springer, 2004, pp. 472–486.

[50] P. Hoffman and P. McManus, "Dns queries over https (doh)," RFC 8484, 2018.

[51] Z. Hu, L. Zhu, and P. H. J. Heidemann, A. Mankin D. Wessels, "Specification for dns over transport layer security (tls)," RFC 7858, 2016.

[52] R. Bortolameotti, A. Peter, M. H. Everts, and D. Bolzoni, "Indicators of malicious ssl connections," in *International Conference on Network and System Security*. Springer, 2015, pp. 162–175.

[53] W. M. Shbair, T. Cholez, A. Goichot, and I. Chrisment, "Efficiently bypassing sni-based https filtering," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 990–995.

[54] D. Wagner, B. Schneier *et al.*, "Analysis of the ssl 3.0 protocol," in *The Second USENIX Workshop on Electronic Commerce Proceedings*, 1996.

[55] H. Cheng and R. Avnur, "Traffic analysis of ssl encrypted web browsing," 1998.

[56] Q. Sun, D. R. Simon, Y.-M. Wang, W. Russell, V. N. Padmanabhan, and L. Qiu, "Statistical identification of encrypted web browsing traffic," in *Security and Privacy*. IEEE, 2002.

[57] A. Hintz, "Fingerprinting websites using traffic analysis," in *International Workshop on Privacy Enhancing Technologies*. Springer, 2002.

[58] L. Lu, E.-C. Chang, and M. C. Chan, "Website fingerprinting and identification using ordered feature sequences," in *Computer Security – ESORICS 2010*, D. Gritzalis, B. Preneel, and M. Theoharidou, Eds. Springer Berlin Heidelberg, 2010.

[59] M. Liberatore and B. N. Levine, "Inferring the source of encrypted http connections," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*. ACM, 2006.

[60] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting : attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 2009.

[61] U. Cambridge, "Introduction to information retrieval," 2009.

[62] R. Dingledine, N. Mathewson, and P. Syverson, "Tor : The second-generation onion router," Naval Research Lab Washington DC, Tech. Rep., 2004.

[63] B. Zantout, R. Haraty *et al.*, "I2p data communication system," in *Proceedings of ICN*. Citeseer, 2011, pp. 401–409.

[64] A. Montieri, D. Ciuonzo, G. Aceto, and A. Pescapé, "Anonymity services tor, i2p, jondonym : Classifying in the dark," in *2017 29th international teletraffic congress (ITC 29)*, vol. 1. IEEE, 2017, pp. 81–89.

[65] S. Burschka and B. Dupasquier, "Tranalyzer : Versatile high performance network traffic analyser," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2016, pp. 1–8.

[66] K. Shahbar and A. N. Zincir-Heywood, "Anon17 : Network traffic dataset of anonymity services," *Faculty of Computer Science Dalhousie University, Tech. Rep*, 2017.

[67] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*. ACM, 2011.

[68] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson, "Touching from a distance : Website fingerprinting attacks and defenses," in *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 2012.

[69] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8, 1966, pp. 707–710.

[70] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Communications of the ACM*, vol. 7, no. 3, pp. 171–176, 1964.

[71] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting." in *USENIX Security Symposium*, 2014.

[72] A. Panchenko, F. Lanze, J. Pennekamp, T. Engel, A. Zinnen, M. Henze, and K. Wehrle, "Website fingerprinting at internet scale." in *NDSS*, 2016.

[73] M. Juarez, S. Afroz, G. Acar, C. Diaz, and R. Greenstadt, "A critical evaluation of website fingerprinting attacks," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014.

[74] J. Hayes and G. Danezis, "k-fingerprinting : A robust scalable website fingerprinting technique." in *USENIX Security Symposium*, 2016.

[75] T. Wang and I. Goldberg, "Comparing website fingerprinting attacks and defenses," Technical Report 2013-30, CACR, 2013. http ://cacr. uwaterloo. ca/techreports . . . , Tech. Rep., 2014.

[76] C. V. Wright, S. E. Coull, and F. Monrose, "Traffic morphing : An efficient defense against statistical traffic analysis." in *NDSS*, vol. 9. Citeseer, 2009.

[77] M. Juárez, M. Imani, M. Perry, C. Dıaz, and M. Wright, "Wtf-pad : toward an efficient website fingerprinting defense for tor," *CoRR, abs/1512.00524*, 2015.

[78] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Peek-a-boo, i still see you : Why efficient traffic analysis countermeasures fail," in *2012 IEEE symposium on security and privacy*. IEEE, 2012, pp. 332–346.

[79] S. Dai, A. Tongaonkar, X. Wang, A. Nucci, and D. Song, "Networkprofiler : Towards automatic fingerprinting of android apps," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 809–817.

[80] A. Tongaonkar, "A look at the mobile app identification landscape," *IEEE Internet Computing*, vol. 20, no. 4, pp. 9–15, 2016.

[81] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic, "Who do you sync you are ? smartphone fingerprinting via application behaviour," in *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, 2013, pp. 7–12.

[82] Q. Wang, A. Yahyavi, B. Kemme, and W. He, "I know what you did on your smartphone : Inferring app usage over encrypted data traffic," in *2015 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2015, pp. 433–441.

[83] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Transactions on Information Forensics and Security*, Jan 2018.

[84] ——, "Appscanner : Automatic fingerprinting of smartphone apps from encrypted network traffic," in *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, 2016.

[85] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Multi-classification approaches for classifying mobile app traffic," *Journal of Network and Computer Applications*, vol. 103, pp. 131–145, 2018.

[86] T. Bakhshi and B. Ghita, "On internet traffic classification : A two-phased machine learning approach," *Journal of Computer Networks and Communications*, vol. 2016, 2016.

[87] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapè, "Mimetic : Mobile encrypted traffic classification using multimodal deep learning," *Computer Networks*, vol. 165, p. 106944, 2019.

[88] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson, "The security impact of https interception." in *NDSS*, 2017.

[89] LRP, LAM, BAI, and MRR, "Recommandations de sécurité concernant l'analyse des flux https," ANSSI, Tech. Rep., 2014.

[90] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox : Deep packet inspection over encrypted traffic," in *ACM SIGCOMM Computer Communication Review*. ACM, 2015.

[91] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 965–976.

[92] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*. IEEE, 2000, pp. 44–55.

[93] Y.-H. Lin, S.-H. Shen, M.-H. Yang, D.-N. Yang, and W.-T. Chen, "Privacy-preserving deep packet filtering over encrypted traffic in software-defined networks," in *Communications (ICC)*. IEEE, 2016.

[94] J. Ning, G. S. Poh, J.-C. Loh, J. Chia, and E.-C. Chang, "Privdpi : Privacy-preserving encrypted traffic inspection with reusable obfuscated rules," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA : Association for Computing Machinery, 2019, p. 1657–1670. [Online]. Available : https://doi.org/10.1145/3319535.3354204

[95] A. Reed and M. Kranch, "Identifying https-protected netflix videos in real-time," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. ACM, 2017.

[96] A. Reed and B. Klimkowski, "Leaky streams : Identifying variable bitrate dash videos streamed over encrypted 802.11 n connections," in *2016 13th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2016, pp. 1107–1112.

[97] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson, "Language identification of encrypted voip traffic : Alejandra y roberto or alice and bob ?" in *USENIX Security Symposium*, 2007.

[98] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson, "Spot me if you can : Uncovering spoken phrases in encrypted voip conversations," in *Security and Privacy*. IEEE, 2008.

[99] K. Norrman, D. McGrew, M. Naslund, E. Carrara, and M. Baugher, "The secure real-time transport protocol (srtp)," RFC, 2004.

[100] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Analyzing android encrypted network traffic to identify user actions," *IEEE Transactions on Information Forensics and Security*, Jan 2016.

[101] J. Liu, Y. Fu, J. Ming, Y. Ren, L. Sun, and H. Xiong, "Effective and real-time in-app activity analysis in encrypted internet traffic streams," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017.

[102] Y. Fu, H. Xiong, X. Lu, J. Yang, and C. Chen, "Service usage classification with encrypted internet traffic in mobile messaging apps," *IEEE Transactions on Mobile Computing*, Nov 2016.

[103] B. Saltaformaggio, H. Choi, K. Johnson, Y. Kwon, Q. Zhang, X. Zhang, D. Xu, and J. Qian, "Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic," in *10th USENIX Workshop on Offensive Technologies*. USENIX Association, 2016.

[104] S. E. Coull and K. P. Dyer, "Traffic analysis of encrypted messaging services : Apple imessage and beyond," *ACM SIGCOMM Computer Communication Review*, 2014.

[105] D. X. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on ssh." in *USENIX Security Symposium*, 2001.

[106] "Traffic analysis on google maps with gmaps-trafficker," white paper, 2012. [Online]. Available : http://blog.ioactive.com/2012/02/ssl-traffic-analysis-on-google-maps.html

[107] A. Kwon, M. AlSabah, D. Lazar, M. Dacier, and S. Devadas, "Circuit fingerprinting attacks : Passive deanonymization of tor hidden services," in *24th {USENIX} Security Symposium ({USENIX} Security 15)*, 2015, pp. 287–302.

[108] F. Mosteller, "Understanding the birthday problem," in *Selected Papers of Frederick Mosteller*. Springer, 2006.

[109] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977. [Online]. Available : http://www.jstor.org/stable/2984875

[110] E. Parzen, "On estimation of a probability density function and mode," *The annals of mathematical statistics*, 1962.

[111] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn : Machine learning in Python," *Journal of Machine Learning Research*, 2011.

[112] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of frameworks for high-performance packet io," in *Symposium on Architectures for Networking and Communications Systems (ANCS)*. ACM/IEEE, 2015.

[113] S. Marchal, J. François, R. State, and T. Engel, "Proactive discovery of phishing related domain names," in *15th International Conference on Research in Attacks, Intrusions, and Defenses (RAID)*. Springer-Verlag, 2012.

[114] I. Grigorik, "Optimizing tls record size & buffering latency," Blog, 2013. [Online]. Available : https://www.igvita.com/2013/10/24/optimizing-tls-record-size-and-buffering-latency

[115] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA : Association for Computing Machinery, 2006, p. 161–168. [Online]. Available : https://doi.org/10.1145/1143844.1143865

[116] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, p. 5–32, Oct. 2001. [Online]. Available : https://doi.org/10.1023/A:1010933404324

[117] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Monterey, CA : Wadsworth and Brooks, 1984.

[118] L. Ceriani and P. Verme, "The origins of the gini index : extracts from variabilità e mutabilità (1912) by corrado gini," *The Journal of Economic Inequality*, vol. 10, no. 3, pp. 421–443, 2012. [Online]. Available : https://EconPapers.repec.org/RePEc:kap:jecinq:v:10:y:2012:i:3:p:421-443

[119] R. Genuer and J.-M. Poggi, "Arbres CART et Forêts aléatoires,Importance et sélection de variables," Jan. 2017, working paper or preprint. [Online]. Available : https://hal.archives-ouvertes.fr/hal-01387654

[120] T. Wang and I. Goldberg, "Improved website fingerprinting on tor," in *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society.* ACM, 2013, pp. 201–212.

[121] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval.* ACM, 2015.

[122] V. Shmatikov and M.-H. Wang, "Timing analysis in low-latency mix networks : Attacks and defenses," in *European Symposium on Research in Computer Security.* Springer, 2006, pp. 18–33.

[123] P.-O. Brissaud, J. François, I. Chrisment, T. Cholez, and O. Bettan, "Transparent and Service-Agnostic Monitoring of Encrypted Web Traffic," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 842–856, Sep. 2019. [Online]. Available : https://hal.inria.fr/hal-02316644

[124] P.-O. Brissaud, J. François, I. Chrisment, T. Cholez, and O. Bettan, "Passive monitoring of https service use," in *14th International Conference on Network and Service Management (CNSM).* IEEE, 2018, pp. 219–225. [Online]. Available : https://hal.archives-ouvertes.fr/hal-01943936

[125] ——, "Encrypted http/2 traffic monitoring : Standing the test of time and space," in *IEEE International Workshop on Information Forensics and Security (WIFS).* IEEE, 2020. [Online]. Available : https://hal.archives-ouvertes.fr/hal-03032578

[126] ——, "Analyse de trafic chiffré : Application au web," in *Rendez-Vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information (RESSI)*, 2018.

Résumé

L'usage du chiffrement pour protéger la vie privée des utilisateurs est devenue la norme pour l'ensemble des services web. Ainsi, il n'est plus possible d'utiliser les outils habituels, tel que l'inspection de paquet, pour détecter des comportements illicites sur Internet. L'enjeu principal de cette thèse est donc de trouver de nouvelles solutions alternatives pour superviser certains comportements utilisateur dans du trafic HTTPS, tout en respectant trois principes : passivité, transparence et respect de la vie privée. Ce besoin est réel car dans le domaine de l'analyse de trafic chiffré, l'état de l'art s'est développé principalement autour de la reconnaissance des protocoles ou des service mais pas sur la détections des actions utilisateur.

Un premier objectif de cette thèse est de superviser des mots-clés recherchés sur un moteur de recherche d'images lors de l'usage de HTTPS couplé avec HTTP/1.1. Cette solution passe par la reconstruction des tailles d'objets HTTP pour construire des signatures du trafic avec la méthode de l'estimation par noyau (KDE). Lors de l'évaluation de cette solution afin de détecter l'utilisation de plus de 10 000 mots-clés sur le service Google Images, notre solution de classification atteint un taux de justesse de plus de 99% en considérant un scénario en monde ouvert. Cette approche convient pour du trafic chiffré HTTP/1.1 mais voit ses performances limités face à du trafic HTTP/2, car cette version de HTTP à un fort impact sur le trafic.

Ainsi, dans un deuxième temps, l'objectif est d'adapter nos connaissances pour réaliser de la détection sur du trafic HTTPS couplé avec HTTP/2. Cette nouvelle solution de supervision, s'articule autour de caractéristiques observables sur du trafic chiffré et adaptées pour HTTP/2. Les caractéristiques couplées à une solution d'apprentissage supervisé (les forêts d'arbres décisionnels) permettent la création d'un modèle de classification. Cette solution nommé H2Classifier est évaluée sur plusieurs services très utilisés (Amazon, Google, Google Images et Google Maps) et affiche un TPR entre 61 et 98% dépendant du service considéré, lors de la supervision de 2000 mots-clés (par service) dans le cadre d'un scénario en monde ouvert. Finalement H2Classifier est également testé dans différentes situations permettant d'évaluer l'impact du temps, des services et des configurations sur ce nouvel outil de supervision.

Mots-clés: Supervision, Trafic chiffré, HTTPS, Analyse de trafic, Détection d'actions utilisateur, Apprentissage supervisé, HTTP/2, HTTP/1.1

Abstract

The protection of the Internets' users privacy has made every web service offer some security by using encryption. Thus, it is now impossible to use classical tools anymore, like DPI (deep pakets inspection), in order to detect malicious behaviour on the Internet. The main target of this thesis is to find new ways to monitor malicious behaviours despite the use of encryption (HTTPS). This new solution should, nevertheless, follow tree guidelines : passivity, transparency and privacy preservation. According to the works in the state of the art for encrypted trafic monitoring, they mainly focus about protocols or services detection but not about the detections of the users' behavior inside a service.

The first objective is to construct a monitoring solution in order to detect some behaviour inside a web service protected by HTTPS used with HTTP/1.1. We developp an example which detects requests related to non legitimate keywords on Images search engine by only monitoring the encrypted traffic. The solution reconstructs the size of the encrypted HTTP objets and builds a footprint of the related trafic by using the Kernel Density estimation method (KDE). The evaluation of this traffic classification when monitoring 10 000 keywords achieved an accuracy of more than 99% considering an open world scenario. Despite, this solution is very effective when monitoring HTTP/1.1 traffic, it show some limitation when dealing with HTTP/2 traffic because of its impact on the traffic.

Thus, the second goal is to adapt our knowledge for purposes of detecting keywords when HTTPS is used with HTTP/2. This new method is structured around some features collected on the encrypted trafic and use supervised machine learning (random forest) to classify them. The solution called H2Classifier is evaluated over four very used services (Amazon, Google, Google Images and Google Maps) and achieve a TPR between 61 and 98% depending of the service when monitoring 2000 keywords (per service) considering open world scenario. Finally H2Classifier evaluated over the time, against new services and with new configurations too.

Keywords: Traffic monitoring, Encrypted Traffic, HTTPS, Traffic analysis, Detecting user actions, KDE, Random Forest, HTTP/2, HTTP/1.1