



**Ecole Polytechnique de l'Université François Rabelais de Tours**

Département Informatique

64 avenue Jean Portalis

37200 Tours, France

Tél. +33 (0)2 47 36 14 14

[polytech.univ-tours.fr](http://polytech.univ-tours.fr)

**Projet Architecture, Système et Réseaux**

**2018-2019**

# **Crawling web et requête HTTP par serveur proxy**

**Tuteur académique**

**Mathieu DELALANDRE**

**Étudiants**

**Thomas COUCHOUD (DI5)**

**Victor COLEAU (DI5)**

9 novembre 2018



## Liste des intervenants

Nom	Email	Qualité
Thomas COUCHOUD	<a href="mailto:thomas.couchoud@etu.univ-tours.fr">thomas.couchoud@etu.univ-tours.fr</a>	Étudiant DI5
Victor COLEAU	<a href="mailto:victor.coleau@etu.univ-tours.fr">victor.coleau@etu.univ-tours.fr</a>	Étudiant DI5
Mathieu DELALANDRE	<a href="mailto:mathieu.delalandre@univ-tours.fr">mathieu.delalandre@univ-tours.fr</a>	Tuteur académique, Département Informatique



# Avertissement

Ce document a été rédigé par Thomas Couchoud et Victor Coleau susnommés les auteurs.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Mathieu Delalandre susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

Les auteurs reconnaissent assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

Les auteurs attestent que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

Les auteurs attestent ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

Les auteurs attestent que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

Les auteurs reconnaissent qu'ils ne peuvent diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

Les auteurs autorisent l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.



## Pour citer ce document

Thomas Couchoud et Victor Coleau, *Crawling web et requête HTTP par serveur proxy*, Projet Architecture, Système et Réseaux, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2018-2019.

```
@mastersthesis{
  author={Couchoud, Thomas and Coleau, Victor},
  title={Crawling web et requête HTTP par serveur proxy},
  type={Projet Architecture, Système et Réseaux},
  school={Ecole Polytechnique de l'Université François Rabelais de Tours},
  address={Tours, France},
  year={2018-2019}
}
```

# Table des matières

Liste des intervenants	a
Avertissement	b
Pour citer ce document	c
Table des matières	i
Table des figures	iii
Liste des tableaux	iv
<b>I Introduction</b>	<b>1</b>
<b>II Veille technique</b>	<b>2</b>
<b>1 Stratégies de défense</b>	<b>3</b>
1 Liste noire .....	3
2 CAPTCHA.....	4
3 Modification du DOM.....	4
4 Attrapes-bots.....	4
<b>2 Outils d'analyse</b>	<b>6</b>
1 Logs .....	6
2 Urchin .....	7

<b>3</b>	<b>Stratégies de masquage</b>	<b>10</b>
1	Comportement du crawler.....	10
2	Utilisation de plusieurs identifiants .....	11
<b>4</b>	<b>Les proxy</b>	<b>12</b>
1	Késako ? .....	12
1.1	Avantages .....	12
1.2	Inconvénients.....	13
2	Alternative - VPN .....	13
3	Présentation de proxys.....	13
<b>III</b>	<b>Application : Réalisation d'une crawler avec proxy</b>	<b>15</b>
<b>5</b>	<b>Crawler basique</b>	<b>16</b>
1	Méthode implémentée .....	16
1.1	Crawlers .....	16
1.2	Downloaders .....	17
2	Résultats de tests .....	17
<b>6</b>	<b>Utilisation de stratégies de masquage</b>	<b>18</b>
1	User agent .....	20
2	Limite de requêtage .....	20
2.1	Sur l'ensemble .....	20
2.2	Par thread .....	20
<b>7</b>	<b>Proxy</b>	<b>22</b>
<b>IV</b>	<b>Conclusion</b>	<b>23</b>
	<b>Annexes</b>	<b>24</b>
<b>A</b>	<b>Acronymes</b>	<b>25</b>



# Table des figures

## Liste des tableaux

1	Crawler .....	1
<b>2</b>	<b>Outils d'analyse</b>	
1	Nombre de visites .....	7
2	Temps de visite.....	8
3	Provenance des visites.....	8
4	Robots/User-Agents .....	9
5	Robots/User-Agents précis.....	9
<b>4</b>	<b>Les proxy</b>	
1	La place d'un proxy dans une communication .....	12
2	La place d'un VPN dans une communication .....	13
<b>6</b>	<b>Utilisation de stratégies de masquage</b>	
1	Paramètres de la requête GET.....	19



# Liste des tableaux

<b>4</b>	<b>Les proxy</b>	
2	Caractéristiques utilisateurs .....	14
<b>5</b>	<b>Crawler basique</b>	
2	Résultats sur différents sites.....	17



## Première partie

# Introduction

Dans notre société moderne l'Internet occupe une place très importante. Il offre une quantité pharaonique d'information en libre accès. Parmi ces sources d'information, on trouve notamment des « wikis » qui sont des encyclopédies collaboratives permettant la large diffusion de données. Malgré leur apparente générosité et leur connaissance des pratiques, ces sites n'apprécient que peu que les données qu'ils fournissent en soient extraites.

Cette nouvelle mane d'information attire les convoitises et polarise les comportements. D'un côté nous retrouvons les collecteurs de données cherchant à en agréger et stocker de plus en plus de leur propre chef. De l'autre les sites mettant à disposition l'information dont le but paradoxal est de fournir gratuitement tout en conservant l'exclusivité.

Cela entraine une guerre technologique entre crawlers et sites web. Les premiers développent des technologies de plus en plus efficaces, rapides et discrètes. Les seconds cherchent à contrecarrer les premiers grâce à des techniques de détection de plus en plus sophistiquées.

Le but de ce projet est d'étudier à la fois les techniques mises en place par les crawler pour se rendre invisible et celles mise en place par les sites pour se défendre. Cette recherche se concrétisera par la réalisation d'un crawler effectuant ses connexions au travers d'un proxy.



Figure 1 – *Crawler*

Deuxième partie

Veille technique

# 1

## Stratégies de défense

Afin de se défendre face aux demandes massives que peuvent représenter les crawlers, les créateurs de sites web ont imaginés plusieurs méthodes de contre-attaque. Dans cette partie nous allons en développer quelques unes.

### 1 Liste noire

Le principe de base d'une liste noire est de bannir du site les « utilisateurs » trop agressifs. Un « utilisateur » peut être un compte inscrit sur le site ou plus simplement une adresse IP requêtant le serveur.

La problématique principale de cette méthode est différencier un utilisateur humain d'un automate. Afin de prendre la décision de bannir ou non, plusieurs moyens sont à disposition des administrateurs :

- User-agent : Le **user-agent (UA)** est un champ renseigné dans l'entête d'une requête HTTP. Son but est d'identifier l'outil qui a engendré cette demande. Par exemple un **UA** de Firefox ressemble à « Mozilla/5.0 (X11 ; Ubuntu ; Linux\_x86\_64 ; rv:62.0) Gecko/20100101 Firefox/62.0 » tandis que celui d'un bot Google est de la forme « Mozilla/5.0 (compatible ; Googlebot/2.1 ; +http://www.google.com/bot.html) ».

Grace à cette identification, il est possible de filtrer les requêtes pour n'accepter que celles provenant des navigateurs web standards.

- Adresse IP : Une approche peut être de se baser sur l'adresse IP permettant d'identifier une machine ou un réseau précis.
- Comportement de l'utilisateur : A partir des méthodes d'identifications précédentes, il est possible de définir une stratégie de bannissement plus juste. En effet bannir tous les navigateurs qui ne sont pas Internet Explorer ou bien toutes les adresses IP commençant par 1 n'est que peu pertinent.

Il est alors possible d'étudier les méthodes d'explorations du site afin de dénicher les comportements indésirables. Ces derniers peuvent être plus ou moins simples à observer.

- Un nombre de requêtes humainement impossible (ex : 5 requêtes par seconde).
- Des temps de visite des pages très courts (ex : 0.5 secondes par page).

- Une fréquence de requêtage régulière (ex : toutes les secondes).
- Les sauts de pages à d'autres non reliées par hyperlien.

La pertinence de cette méthode de bannissement dépend grandement de la qualité de reconnaissance des automates. En effet, il se peut qu'un automate imite parfaitement le comportement humain, et donc ne soit pas détecté, tout comme un comportement humain inhabituel pourrait être banni par erreur. L'objectif est donc de développer des méthodes de reconnaissance adaptées afin de ne pas perdre des visiteurs désirés.

## 2 CAPTCHA

Le CAPTCHA est une famille de tests de Turing ayant pour but de différencier un utilisateur humain d'un automate. Plusieurs types de CAPTCHAs sont imaginables :

- Reconnaître une suite de lettres altérées (ex : déformées, barrées, avec des trous, ...).
- Cocher une case qui vérifie le comportement précédent de l'utilisateur.
- Reconnaître des parties spécifiques d'une image.



## 3 Modification du DOM

Une autre approche serait non plus d'essayer de rejeter les automates mais de leur compliquer la tâche.

Une méthode possible consiste à modifier constamment et régulièrement la structure du site (nom des classes, IDs, ...). De même, il est envisageable de remplacer certaines parties du site (notamment le texte) par des images.

De ce fait les automates spécifiques à un site donné devront être réadaptés fréquemment ce qui peut décourager leur développeur.

## 4 Attrapes-bots

Afin de distinguer un humain d'un bot, il serait envisageable de baser notre différenciation sur l'exploration ou non d'une page web normalement non-visible d'un utilisateur lambda. Html

/ CSS offrent la possibilité de masquer certains éléments tout en les incluant au sein du code source de la page.

Cependant, un crawler inattentif ne fera pas la différence entre des éléments visibles ou non. On pourrait donc renforcer la sécurité du site grâce à des liens pigés et bannir toute IP s'y rendant.

# 2

## Outils d'analyse

Lors de la réalisation de notre crawler et des stratégies de masquage qu'il implémente, nous avons dû analyser les traces qu'il laisse sur un site. Pour cela nous avons mis en place un serveur apache et lancé notre crawler dessus.

Nous allons ici présenter deux manières d'analyser l'activité des clients d'un serveur.

### 1 Logs

La manière la plus simple de voir le trafic de notre site est de regarder les logs.

L'exemple ci-dessus est un extrait de l'un de ces logs. On y retrouve les informations suivantes :

- Adresse IP du client.
- Le nom de domaine.
- L'heure.
- Le type de requête (GET, POST, ...).
- La page demandée.
- Le protocole (HTTP/1.1).
- Le code HTTP retourné (200, 404, 403, 500, ...).
- La taille de la réponse.
- Origine de la requête.
- UA.

Grâce à celles-ci nous pouvons identifier deux des caractéristiques citées plus haut (UA et IP) du client.

Nous pouvons, à partir d'un tel fichier, induire des informations plus poussées. Par exemple il est possible de recréer le chemin parcouru par l'utilisateur grâce à son adresse IP, UA et temps de la requête.

Du fait de sa grossièreté, le fichier de log est peu pratique à lire et interpréter. En revanche, un algorithme performant pourra repérer les utilisations incohérentes, impossibles et/ou humainement infaisables.

## 2 Urchin

Afin de simplifier l'analyse des logs pour tout le monde, des outils spécifiques ce sont créés. Nous allons ici présenter Urchin v6 qui analyse les logs et offre des vues graphiques de nombreuses métriques. Ces dernières peuvent être obtenues sur différentes périodes (journée, mois, période définie, ...).

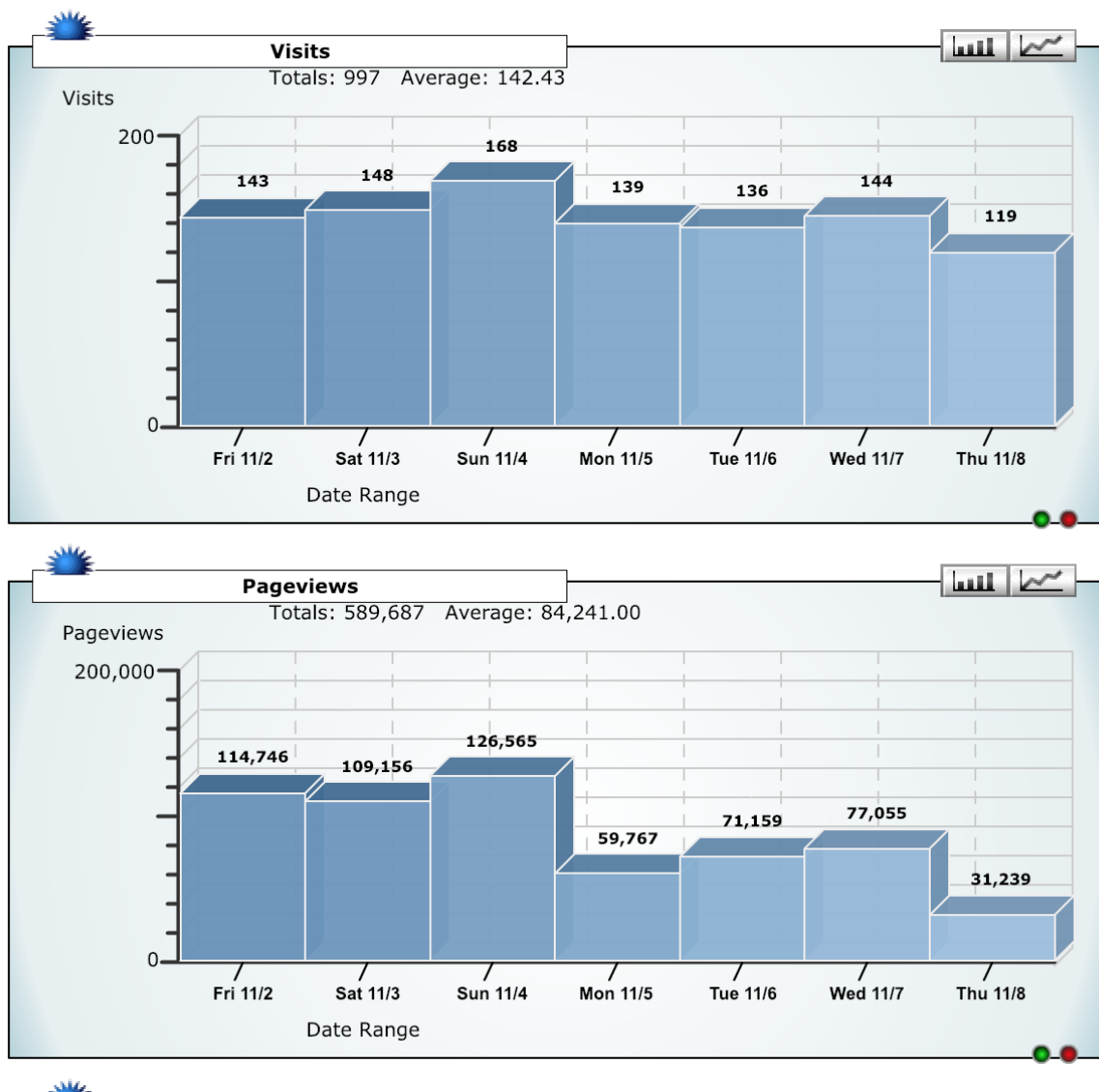
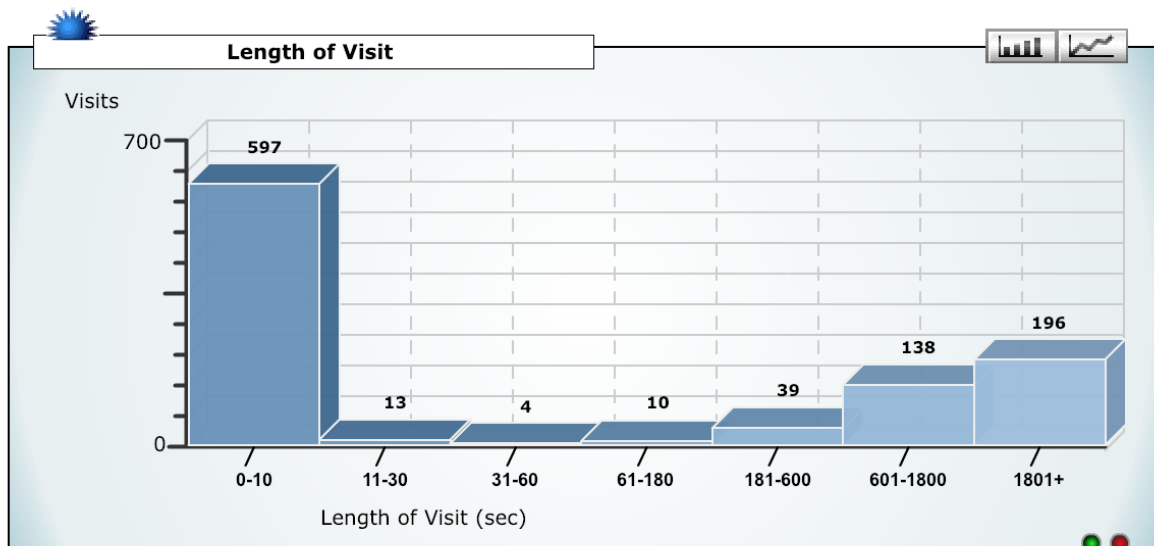


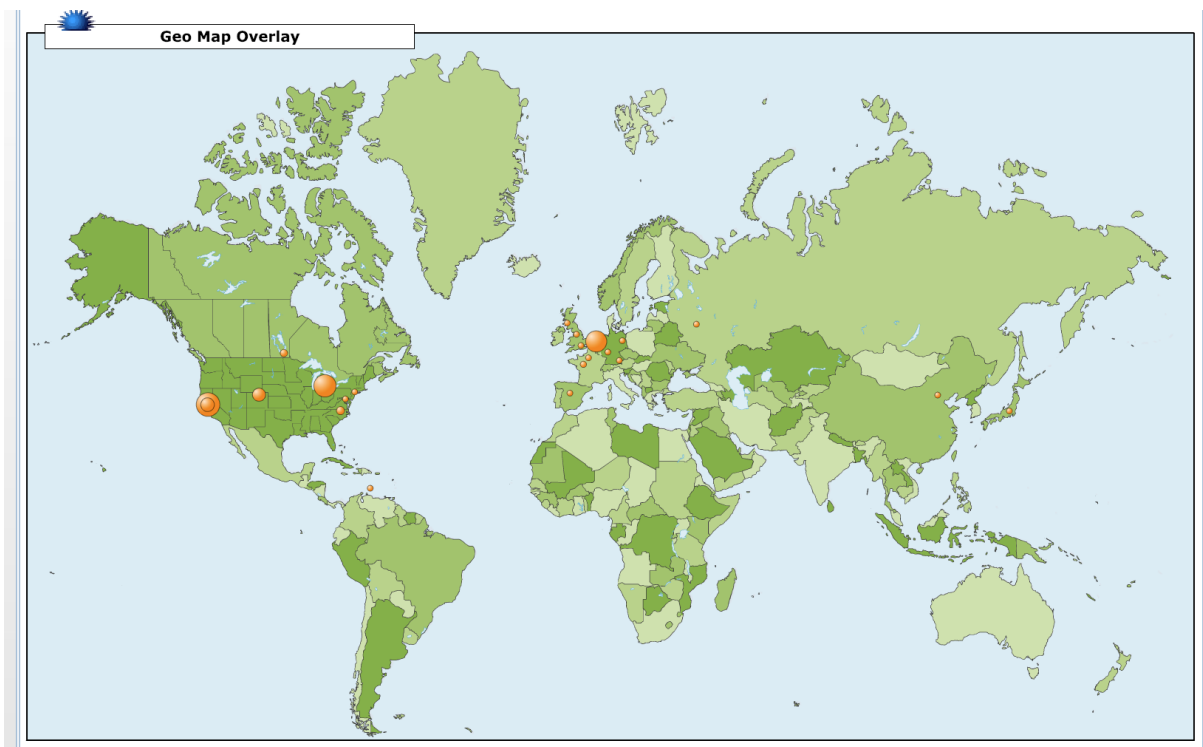
Figure 1 – Nombre de visites

Sur ces représentations graphiques nous pouvons facilement analyser les pics de fréquentation. Ceux-ci peuvent être associés à l'arrivée d'un bot.



**Figure 2** – *Temps de visite*

Ce schéma nous illustre les temps de visites par page. Si un pic entre 0-10 secondes se fait ressentir, il est possible que cela soit dû à un automate.



**Figure 3** – *Provenance des visites*

Dans le cas d'un site localisé (une langue précise par exemple), il peut être utile d'obtenir les origines des requêtes. Si un grand nombre sont effectuées depuis un pays extérieur, il est peu probable que cela soit normal.



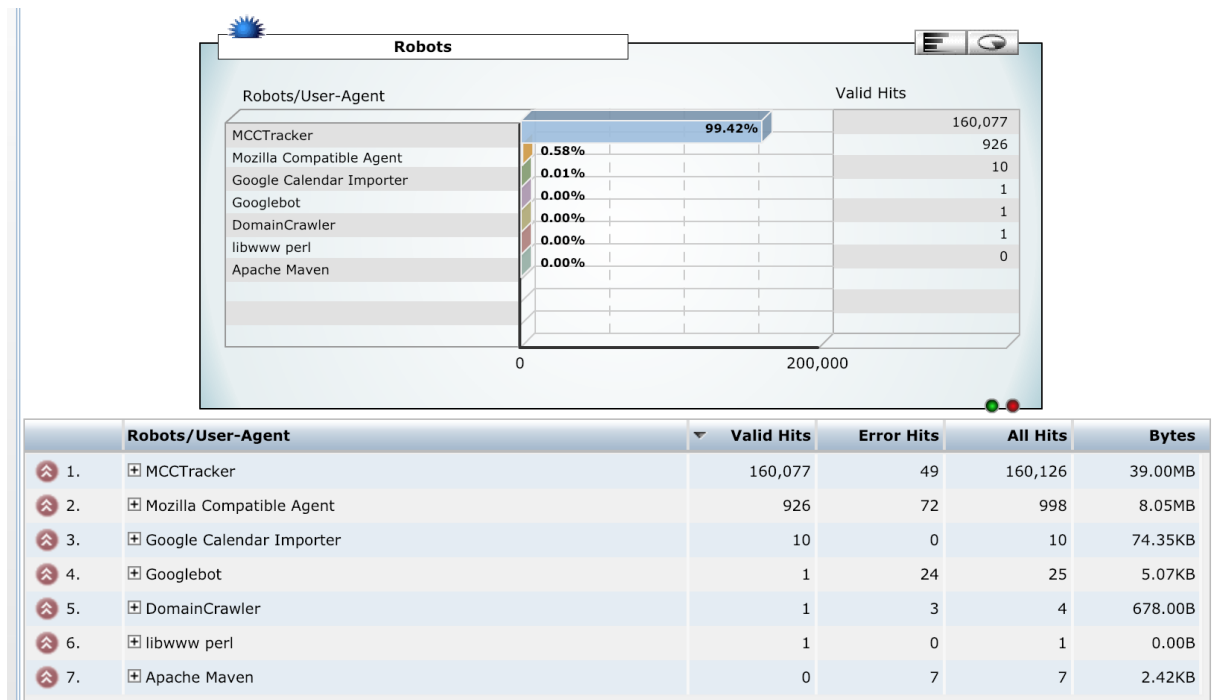


Figure 4 – Robots/User-Agents

	Robots/User-Agent	Valid Hits	Error Hits	All Hits	Bytes
	<input type="checkbox"/> Googlebot	1	24	25	5.07KB
1.	Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot	1	24	25	5.07KB

Figure 5 – Robots/User-Agents précis

Sur cette page nous pouvons observer les **UAs** qui nous ont visités. Si l'on en repère un étrange et récurrent, il faut alors se poser des questions. Il est aussi possible d'obtenir plus de détails sur chaque robot.

# 3

## Stratégies de masquage

Comme décrit précédemment, les gestionnaires de sites web tentent constamment de bloquer l'accès à leur contenu au travers de diverses méthodes. Ces dernières reposent principalement sur le blocage d'un identifiant unique à la personne suite au repérage d'un comportement suspect. Les personnes attaquant les sites ont donc pour but de contourner ces mesures, soit en ayant plusieurs identifiants à leur disposition soit en essayant d'adopter un comportement proche d'un utilisateur normal.

### 1 Comportement du crawler

Une première mesure qui pourrait être envisagée afin d'éviter que le crawler soit identifié est de modifier les headers de requête HTTP par défaut mis en place par les bibliothèques permettant d'effectuer des demandes HTTP. En effet, si l'on prend par exemple la bibliothèque `urllib` de Python, on remarque que le **UA** par défaut est « Python-urllib/3.4 » et que le `Accept-Encoding` est « identity ». Avec un tel **UA**, le modérateur peut facilement comprendre qu'un programme Python est à l'origine de ces requêtes et non un utilisateur humain utilisant un navigateur web standard. Il serait alors plus judicieux de remplacer notre **UA** par celui d'un navigateur web répandu (Chrome, Firefox, Edge, etc.).

De même, le `Accept-Encoding` peut, dans certains cas, compromettre l'anonymat du crawler. Cependant, la modification de certaines informations du header peut entraîner des changements sur le site en question. Par exemple, le header `Accept-Language` indique les préférences utilisateur concernant la langue. Il se peut donc qu'un site mette à disposition deux versions en fonction de la langue souhaitée.

Une deuxième méthode serait de prendre en charge les cookies au travers de JavaScript. En effet, ceux-ci peuvent être utilisés afin de reconnaître un utilisateur ayant visité le site peu de temps auparavant. Si un crawler se rend sur le site de manière intensive sans présenter ce cookie, le serveur pourrait se rendre compte qu'il ne s'agit pas d'un utilisateur faisant des requêtes successives.

Cependant, enregistrer les cookies, peut aussi jouer en notre défaveur : ceux-ci sont une façon de garder une trace d'un utilisateur précis et donc de l'identifier (Voir [Section 1](#) (Chapitre 1)).

Enfin, un trop grand nombre de requêtes en très peu de temps trahit un comportement robotique. Il est donc important de contrôler la vitesse de notre crawler afin de ne pas surcharger la bande passante du site et risquer de se faire bannir.

Cette recommandation est en opposition avec les techniques de parallélisation souvent mises en place afin d'accélérer la vitesse et l'efficacité du crawler.

### 2 Utilisation de plusieurs identifiants

Comme vu précédemment, les principales techniques anti-crawler reposent sur le bannissement d'adresse IP irrespectueuses. Une méthode de contournement basique est donc de changer régulièrement d'adresse IP. Les proxys sont donc une alternative parfaite (Voir [Chapitre 4](#)).

De plus, certains sites utilisent des sessions ou comptes utilisateurs. Un crawler pourrait donc en tirer parti en s'identifiant officiellement sur le site afin de se faire passer pour un utilisateur standard.

Il est d'ailleurs possible de créer et d'utiliser plusieurs comptes en parallèle afin de répartir la charge de requêtes.

# 4

## Les proxy

### 1 Késako ?

Un proxy est un composant logiciel servant d'intermédiaire entre deux hôtes afin de faciliter ou de surveiller leurs échanges.

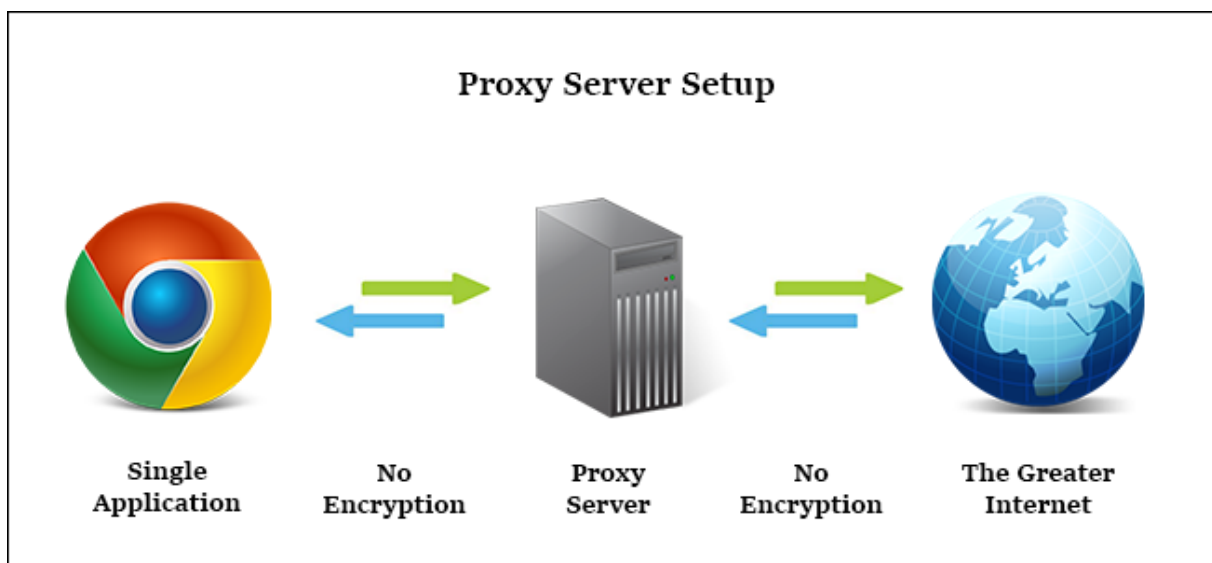


Figure 1 – La place d'un proxy dans une communication

### 1.1 Avantages

Les proxys peuvent par exemple servir à contourner certains filtrages. Supposons le cas d'un pays qui bloque l'accès à certains sites, en se connectant à un proxy non bloqué, l'utilisateur pourra accéder à son site au travers de ce dernier car le proxy ne dispose pas des mêmes règles de filtrage.

A l'inverse, certains établissements scolaires ou entreprises limitent l'accès à certains sites grâce à un serveur proxy. En effet, toutes les requêtes effectuées par les utilisateurs du réseau passe par ce serveur intermédiaire qui bloque les sites dont l'adresse a été spécifiquement interdite.

Un autre avantage de l'utilisation d'un proxy est de pouvoir surfer anonymement. Les sites visités n'ont conscience que de l'adresse du proxy et non de(s) utilisateur(s) caché(s) derrière.

De plus un proxy permet le masquage de son lieu de connexion. En effet le proxy peut ne pas être situé dans le même pays que l'utilisateur. Si un site se base sur un système de géolocalisation pour afficher son contenu (YouTube, Google, Google Maps, etc.), sera prise en compte la géolocalisation du proxy.

## 1.2 Inconvénients

Bien que les proxys offrent de nombreux avantages, ils ont aussi certains inconvénients. S'agissant d'une plateforme reliant un utilisateur au web et effectuant les requêtes du premier, celui-ci voit passer tous les échanges. Certains proxys pourraient alors les enregistrer à des fins malveillantes. L'intérêt d'un proxy étant important, il centralise toutes les requêtes d'une structure et peut donc être saturé ralentissant par la même occasion la connexion de tous les utilisateurs. De manière générale on peut dire qu'une connexion internet passant par un proxy sera toujours plus lente qu'une connexion directe.

## 2 Alternative - VPN

VPN est un acronyme signifiant « Virtual Private Network ». Tout comme les proxys ils permettent de faire apparaître notre navigation internet comme provenant d'une adresse IP distante.

A la différence d'un proxy qui se configure par application au cas par cas (par exemple dans Firefox on peut avoir une configuration différente de celle dans Chrome). En revanche un VPN capture l'ensemble des échanges réseau de la machine et est configuré directement dans le système d'exploitation. Les différentes applications de la machine n'ont donc pas conscience de cette subtilité.

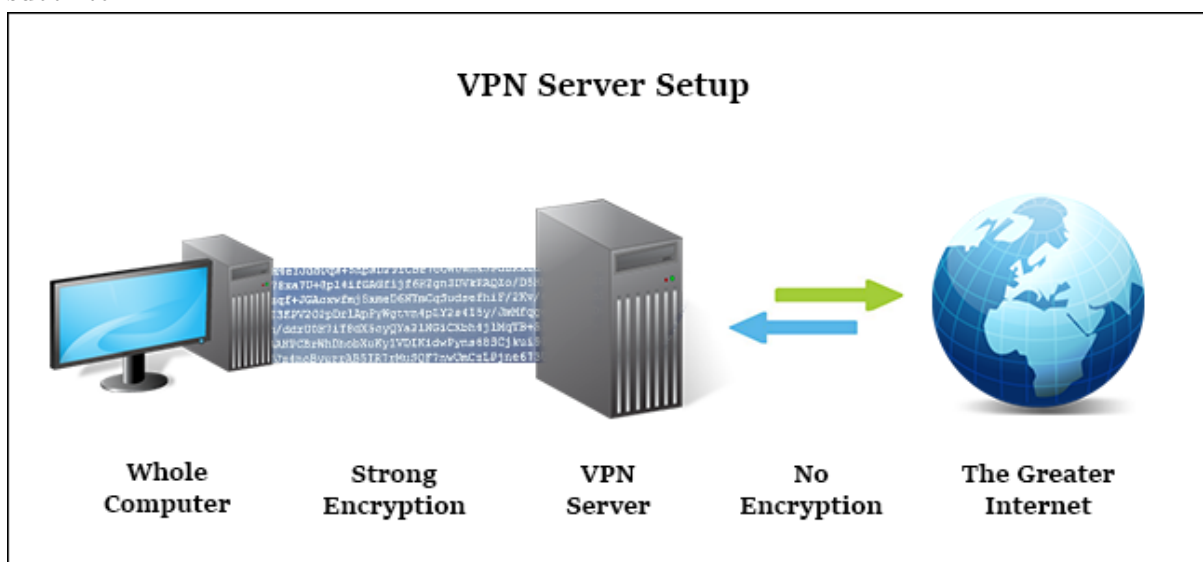


Figure 2 – La place d'un VPN dans une communication

Dans le cas de notre crawler, il paraît plus adapté d'utiliser un proxy. En effet seul ce dernier requiert d'être « masqué » et passer toute notre connexion au travers d'un VPN semble être un peu overkill.

## 3 Présentation de proxys

Nom	Prix	Pays	Avantages	Inconvénients
Proxy6	1.25\$ par IPv4 par mois	Un choix très large	API pour développeurs & Le moins cher	
BuyProxies	2\$ par serveur proxy par mois	Un choix très large	Bande passante illimité & renouvellement mensuel des proxys	
FoxyProxy	10\$ par IP par mois	-		Pays inconnu & Le plus cher
ProxyRack	40\$ pour 50 connexions simultanées par mois	Un choix moyen de pays	Rotation des adresses IP & Bande passante illimitée	Premier pack cher

Table 2 – Caractéristiques utilisateurs

## Troisième partie

# Application : Réalisation d'une crawler avec proxy

# 5

## Crawler basique

Dans le cadre de ce projet nous devons tester les stratégies de masquage adoptables par un crawler. Afin de maîtriser au mieux ces tests, nous avons souhaité développer nous même un tel logiciel.

Afin d'obtenir un résultat de crawling pertinent tout en restant simple d'implémentation, nous avons choisi de se concentrer sur la recherche et téléchargement des images. Pour ce faire, notre crawler explore tous les liens d'une page ainsi que toutes les images. Il téléchargera les images trouvées tandis qu'il continuera d'explorer les liens d'un même domaine.

Dans l'optique de garder le crawler simple, le javascript n'est pas supporté et seuls les liens dans les balises « a » et « img » sont traités.

### 1 Méthode implémentée

Dans un premier temps, les problématiques de blocage ne seront pas prises en compte, seule l'efficacité prime. Pour cela, nous avons pensé à un système supportant le multi-threading augmentant la capacité d'acquisition des données.

Le programme a été découpé en deux parties : les crawlers et les downloaders.

Les crawlers sont en charge de l'exploration du site web et remplissent deux queues. La première contient les prochains liens à explorer et la seconde les liens des images à télécharger.

Les downloaders, eux, ne font que lire les liens de la queue qui leur est associée et télécharge les images.

L'utilisation de ces queues nous permet de séparer l'exploration et le téléchargement dans des threads différents mais aussi d'avoir plusieurs crawlers et plusieurs downloaders.

#### 1.1 Crawlers

Le travail d'un crawler se décompose de la manière suivante :

- Acquisition d'un lien depuis la queue des liens. Si aucun lien n'est disponible, le thread s'endort temporairement et reprendra au début.
- Ajout de ce lien à la liste des liens déjà explorés.



- Téléchargement de la page HTML.
- Lecture de cette page et récupération des différents liens des balises « a » et « img ».
- Tri des liens obtenus :
  - Si le lien correspond à une image (basé sur l'extension) : on vérifie que cette image n'a pas déjà été téléchargée. Si tel est le cas, on l'ajoute dans la queue des images (queue sans doublons).
  - Sinon : il vérifie si le lien a déjà été exploré. Si ce n'est pas le cas on vérifie qu'il fait partie du même domaine que le lien en cours d'exploration. Si tel est le cas on l'ajoute dans la queue des liens à explorer (queue sans doublons).

## 1.2 Downloaders

Le travail d'un downloader se décompose de la manière suivante :

- Acquisition d'un lien depuis la queue des images. Si aucun lien n'est disponible, le thread s'endort temporairement et reprendra au début.
- Ajout de ce lien à la liste des images déjà téléchargées.
- Récupération du nom de l'image à partir du lien.
- Acquisition du contenu :
  - Si le fichier de sortie est déjà présent, on ne fait que renseigner le lien dans un fichier texte au nom de l'image (contiendra tous les liens menant potentiellement à cette dernière).
  - Sinon, on télécharge l'image.

## 2 Résultats de tests

Site	Observations
Wikipedia	
Qwertee	
Etam	
Google	
4chan	
Tinder	
PrettyLittleThings	
LaDechetterieDuWeb	
Qwertee	

Table 2 – Résultats sur différents sites

# 6

## Utilisation de stratégies de masquage

Afin de faire ressembler au mieux les requêtes de notre crawler à celles d'un navigateur standard nous avons décidé d'imiter le plus fidèlement possible leur header. Pour cela nous commençons par inspecter les données que reçoit le serveur suite à la navigation depuis Firefox.

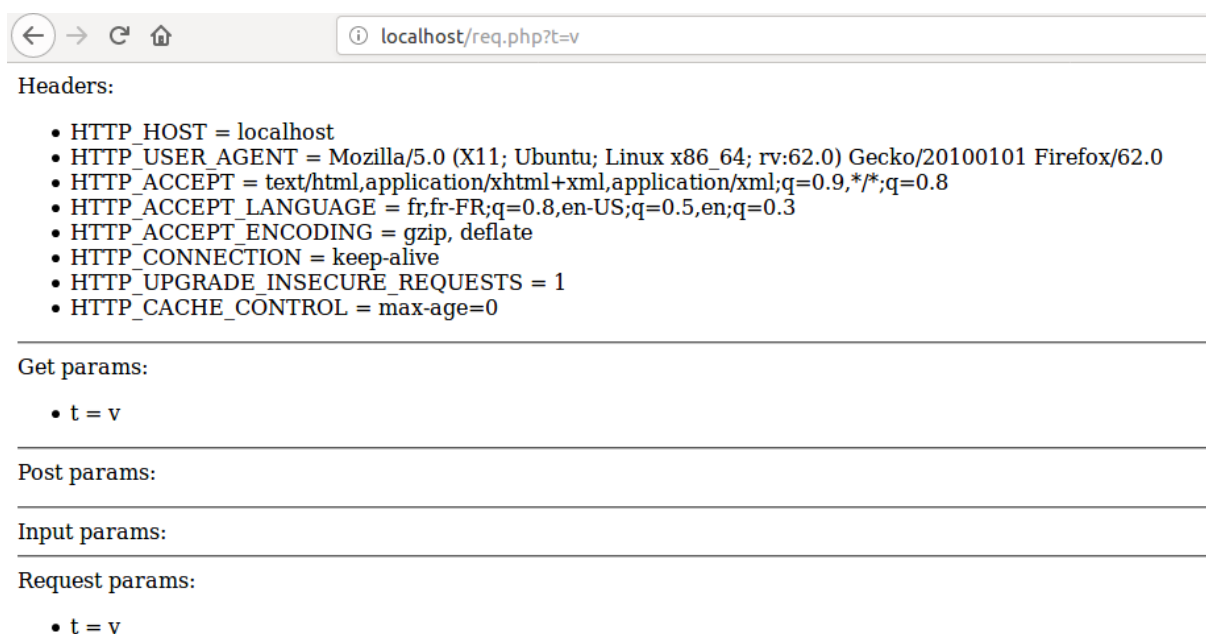
```
1 <html>
2 <head>
3 </head>
4 <body>
5 <div>
6     Headers:
7     <ul>
8     <?php
9         foreach($_SERVER as $h => $v)
10         {
11             if(preg_match('/^HTTP_(.+)/', $h, $hp))
12             {
13                 echo "<li>$h = $v</li>";
14             }
15         }
16     ?>
17     </ul>
18 </div>
19
20 <hr/>
21
22 <div>
23     Get params:
24     <ul>
25     <?php
26         foreach($_GET as $h => $v)
27         {
28             echo "<li>$h = $v</li>";
29         }
30     ?>
31     </ul>
32 </div>
33
34 <hr/>
35
36 <div>
```

```

37     Post params:
38     <ul>
39     <?php
40         foreach($_POST as $h => $v)
41         {
42             echo "<li>$h = $v</li>";
43         }
44     ?>
45     </ul>
46 </div>
47
48 <hr/>
49
50 <div>
51     Input params:
52     <?php
53         echo file_get_contents("php://input");
54     ?>
55 </div>
56
57 <hr/>
58
59 <div>
60     Request params:
61     <ul>
62     <?php
63         foreach($_REQUEST as $h => $v)
64         {
65             echo "<li>$h = $v</li>";
66         }
67     ?>
68     </ul>
69 </div>
70 </body>
71 </html>

```

Avec ce code nous obtenons le résultat suivant :



← → ↻ 🏠 localhost/req.php?t=v

Headers:

- HTTP\_HOST = localhost
- HTTP\_USER\_AGENT = Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:62.0) Gecko/20100101 Firefox/62.0
- HTTP\_ACCEPT = text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8
- HTTP\_ACCEPT\_LANGUAGE = fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3
- HTTP\_ACCEPT\_ENCODING = gzip, deflate
- HTTP\_CONNECTION = keep-alive
- HTTP\_UPGRADE\_INSECURE\_REQUESTS = 1
- HTTP\_CACHE\_CONTROL = max-age=0

---

Get params:

- t = v

---

Post params:

---

Input params:

---

Request params:

- t = v

Figure 1 – Paramètres de la requête GET

## 1 User agent

La première chose a été de changer le **UA**. Etant donné que nous utilisons une librairie (Unirest), il nous suffit de passer en paramètre nos headers. Ces derniers sont basés sur les tests précédents :

- Le **UA**
- Accept
- Accept-Encoding
- Accept-Language
- Cache-Control

Après un test réalisé en local, nous avons pu confirmer que les headers parviennent bien au serveur et le User-Agent est bien celui défini dans le code.

## 2 Limite de requêtage

La deuxième étape consiste à limiter le nombre de requêtes envoyées au serveur. Nous avons imaginé deux méthodes : limiter chaque thread ou limiter l'ensemble de ces derniers.

Dans les deux cas nous faisons une surcharge de `ConcurrentLinkedQueue` et n'avons plus qu'à gérer le délais d'accès. En effet la classe mère gère déjà les accès concurrent.

### 2.1 Sur l'ensemble

Cette classe s'intitule `LimitedGlobalConcurrentLinkedQueue`. Dans celle-ci nous avons redéfini la méthode `poll` permettant d'obtenir le prochain élément.

Le procédé est le suivant :

- On calcule le  $\Delta t$  séparant le temps de notre demande d'accès et le temps du dernier accès à une valeur.
- Si ce  $\Delta t$  est inférieur à un  $\Delta_r$  de référence on endort le thread pour une durée de  $\Delta_r - \Delta t$ . Ce temps écoulé, le thread réitérera sa demande.
- Si  $\Delta t$  est supérieur à  $\Delta_r$  on met à jour le temps du dernier accès et donnons le prochain élément.

Grâce à cette implémentation nous arrivons à passer plus inaperçu auprès du serveur crawlé. Cependant on induit un temps d'attente global très fort ralentissant *de facto* l'exécution de notre programme.

### 2.2 Par thread

Cette classe s'intitule `LimitedThreadConcurrentLinkedQueue`. Dans celle-ci nous avons redéfini la méthode `poll` permettant d'obtenir le prochain élément.

Le procédé est le suivant :

- On calcule le  $\Delta t$  séparant le temps de notre demande d'accès et le temps du dernier accès du thread en cours à une valeur.

- Si ce  $\Delta t$  est inférieur à un  $\Delta_r$  de référence on endort le thread pour une durée de  $\Delta_r - \Delta t$ .
- A son réveil (si il y eut), on met à jour le temps du dernier accès et donnons le prochain élément.

En comparaison à la méthode globale, celle-ci est largement plus rapide mais de fait moins discrète.

7

Proxy

Quatrième partie

Conclusion

## Annexes



# A

## Acronymes

**UA** user-agent. 3, 6, 9, 10, 20

# Crawling web et requête HTTP par serveur proxy

**Résumé**

**Mots-clés**

**Abstract**

**Keywords**

**Tuteur académique**

Mathieu DELALANDRE

**Étudiants**

Thomas COUCHOUD (DI5)

Victor COLEAU (DI5)