**Serveur Proxy Java HTTP/HTTPS**

**Le serveur proxy**

Un serveur proxy est un serveur qui se situe entre le client et le serveur distant sur lequel le client souhaite récupérer des fichiers. Tout le trafic provenant du client est envoyé au serveur proxy, qui adresse des demandes au serveur distant au nom du client. Une fois que le serveur proxy a reçu les fichiers requis, il les transmet au client. Cela peut être bénéfique car cela permet à l'administrateur du serveur proxy de contrôler ce que les machines de son réseau peuvent faire. Par exemple, certains sites Web peuvent être bloqués par le serveur proxy, ce qui signifie que les clients ne pourront pas y accéder. Il est également avantageux que les pages fréquemment visitées puissent être mises en cache par le serveur proxy. Cela signifie que lorsque le client (ou d'autres clients) fait des demandes ultérieures pour des fichiers qui ont été mis en cache, le proxy peut leur fournir les fichiers directement, sans avoir à les demander au serveur distant, ce qui peut être beaucoup plus rapide si le proxy et les clients sont sur le même réseau. Bien que l'on sache que ces fichiers sont contenus dans le cache du proxy, il convient de noter que les clients n'en ont pas connaissance et peuvent maintenir leurs propres caches locaux. L'avantage du cache du proxy est que lorsque plusieurs clients utilisent le proxy, les pages mises en cache par un client peuvent être consultées par un autre client.

**L'implémentation**

Le proxy a été implémenté en utilisant Java et a fait un usage intensif des sockets TCP. Firefox a été configuré pour émettre tout son trafic vers le port et l'adresse IP spécifiés, qui ont ensuite été utilisés dans la configuration du proxy. L'implémentation comprend deux composants principaux : la classe Proxy et la classe RequestHandler. La classe Proxy est responsable de la création d'un ServerSocket qui peut accepter les connexions socket entrantes du client. Cependant, il est vital que l'implémentation soit multithread car le serveur doit être capable de servir plusieurs clients simultanément. Ainsi, dès qu'une connexion socket arrive, elle est acceptée et le proxy crée un nouveau thread qui traite la demande (voir la classe RequestHandler). Comme le serveur n'a pas besoin d'attendre que la demande soit entièrement traitée avant d'accepter une nouvelle connexion socket, les demandes de plusieurs clients peuvent être traitées de manière asynchrone.

La classe Proxy est également responsable de l'implémentation des fonctionnalités de mise en cache et de blocage. En d'autres termes, le proxy est capable de mettre en cache les sites demandés par les clients et d'empêcher dynamiquement les clients de visiter certains sites Web. Comme la vitesse est de la plus haute importance pour le serveur proxy, il est souhaitable de stocker les références aux sites actuellement bloqués et aux sites contenus dans le cache dans une structure de données dont le temps de consultation d'ordre constant est attendu. Pour cette raison, un HashMap a été choisi. Il en résulte des temps de consultation du cache et des sites bloqués extrêmement rapides. Il n'en résulte qu'une faible surcharge si le fichier n'est pas contenu dans le cache, et une augmentation des performances si le fichier était contenu dans le cache. Enfin, la classe proxy est également responsable de la fourniture d'un système de gestion de console dynamique. Cela permet à un administrateur d'ajouter ou de supprimer des fichiers dans le cache et des sites Web dans la liste noire, en temps réel.

**La classe RequestHandler**

La classe RequestHandler est responsable de la gestion des demandes qui arrivent au proxy. Le RequestHandler examine la demande reçue et la traite de manière appropriée. Les demandes peuvent être subdivisées en trois catégories principales : les demandes HTTP GET, les demandes HTTP GET de fichiers contenus dans le cache et les demandes HTTPS CONNEC.

**HTTP GET**

Il s'agit des demandes standard effectuées lorsqu'un client tente de charger une page Web. Le traitement de ces demandes est une tâche simple :

-Extraire l'URL associée à la demande.

-Créer une connexion HTTP à cette URL.

-Écho de la requête GET du client vers le serveur distant.

-Renvoyer la réponse du serveur au client. -Sauvegarder une copie locale du fichier dans le cache du proxy.

**Demande GET HTTP pour un fichier dans le cache**

Comme précédemment, il s'agit des requêtes typiques faites par les clients, mais dans ce cas, le fichier est contenu dans le cache du proxy.

-Extraire l'URL associée à la demande -Hacher l'URL et l'utiliser comme clé pour la structure de données HashMap.

-Ouvrez le fichier résultant pour la lecture.

-Écho du contenu du fichier vers le client.

-Fermez le fichier.

**HTTPS - Demandes de connexion**

Les connexions HTTPS utilisent des sockets sécurisés (SSL). Les données transférées entre le client et le serveur sont cryptées. Cette technique est largement utilisée dans le secteur financier afin de garantir la sécurité des transactions, mais elle est de plus en plus répandue sur Internet. Cependant, à première vue, cela pose un problème pour les serveurs proxy : Comment le proxy va-t-il savoir quoi faire avec ces données cryptées provenant du client ? Afin de surmonter ce problème, dans un premier temps, un autre type de requête HTTP est effectué par le client, une requête CONNECT. Cette requête est une requête HTTP standard, donc non chiffrée, et contient l'adresse de la personne avec laquelle le client veut créer une connexion HTTPS, qui peut être extraite par le proxy. Il s'agit d'un processus connu sous le nom de HTTP Connect Tunneling et fonctionne comme suit :

-Le client émet une demande de connexion. Le proxy extrait l'URL de destination.

-Le proxy crée une connexion socket standard avec le serveur distant spécifié par l'URL.

-En cas de succès, le proxy envoie une réponse '200 Connection Established' au client, indiquant que le client peut maintenant commencer à transmettre les données cryptées au proxy.

-Le proxy transmet alors simultanément toutes les données qui lui sont envoyées par le client vers le serveur distant, et toutes les données reçues du serveur distant vers le client.

Toutes ces données sont cryptées et le proxy ne peut donc pas les mettre en cache ni même les interpréter.