UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et de génie informatique

Infrastructure distribuée permettant la détection d'attaques logicielles

Mémoire de maîtrise
Spécialité : génie électrique

Sébastien DENEAULT

Jury : Frédéric MAILHOT (directeur)
Philippe MABILLEAU
Guy LÉPINE

Sherbrooke (Québec) Canada Aout 2013

Library and Archives Canada

Published Heritage Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et Archives Canada

Direction du Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-96232-9

Our file Notre référence
ISBN: 978-0-494-96232-9

RÉSUMÉ

Le nombre de systèmes informatiques augmente de jour en jour et beaucoup d'entités malveillantes tentent d'abuser de leurs vulnérabilités. Il existe un fléau qui fait rage depuis quelques années et qui cause beaucoup de difficultés aux experts en sécurité informatique : les armées de robots (botnets). Des armées d'ordinateurs infectés sont constituées pour ensuite être louées et utilisées à des fins peu enviables. La société fait face à un problème : il est très difficile d'arrêter ces armées et encore plus de trouver leurs coordonnateurs. L'objectif de ce travail de recherche est de développer des outils destinés à identifier ces entités et aider à démanteler ces réseaux. Plus précisément, ce projet porte sur la conception d'une plateforme distribuée permettant de faire un pré-traitement des données collectées sur divers réseaux et de les distribuer dans un système d'analyse. Cette plateforme sera en libre source, facilement adaptable et flexible. De plus, elle devra être en mesure de traiter une grande quantité de données dans un court laps de temps. Ce système se distinguera étant donné qu'il sera distribué sur plusieurs réseaux sous un modèle client-serveur et collaborera dans le but de trouver les coordonnateurs de ces armées de robots.

Mots-clés : Détection de botnets, Sécurité informatique, Système distribué

REMERCIEMENTS

J'aimerais remercier M. Frédéric Mailhot pour les conseils, le support et l'encadrement qu'il m'a apportés tout au long de ce projet.

De plus, j'aimerais remercier Marc Mazuhelli du Service des Technologies de l'Information de l'Université de Sherbrooke qui a fourni les données de tests pour valider le fonctionnement du système mis en oeuvre lors de ce projet.

Finalement, j'aimerais remercier mes parents Lyne et Gilles, mon frère Julien ainsi que ma copine Marie-Hélène pour le soutien moral qu'ils m'ont accordé tout au long de ce projet.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ACRONYMES

Acronyme	Définition
AMQP	Advanced Message Queuing Protocol
Go	Gigaoctet
HTTP	Hypertext Transfer Protocol
IRC	Internet Relay Chat
JMS	Java Message Service
Mo	Mégaoctet
P2P	Poste-à-poste (peer-to-peer en anglais)
RPM	Tours par minute
SDK	Trousse de développement logicielle
STI	Service des Technologies de l'Information
UdeS	Université de Sherbrooke
XML	Langage de balisage extensible

CHAPITRE 1

INTRODUCTION

1.1 Mise en contexte et problématique

L'informatique est maintenant présente partout, des ordinateurs personnels aux téléphones cellulaires, en passant par les guichets bancaires. Du côté des ordinateurs personnels, la majorité des utilisateurs travaille avec un système d'exploitation Microsoft [47].

Ce système d'exploitation représente un marché potentiel considérable, alors la pression pour déployer rapidement de nouveaux logiciels est très grande. Ceci peut engendrer la mise en marché d'un produit qui a des lacunes de sécurité. Dans cet environnement, le rôle des experts en sécurité informatique est de trouver ces vulnérabilités, de les sécuriser et de faire en sorte qu'elles ne se reproduisent plus. Outre ces lacunes de conception, un enjeu très important est la sensibilisation des utilisateurs de systèmes informatiques, puisqu'une mauvaise utilisation d'un système peut le rendre vulnérable à des attaques.

Des personnes et entreprises malveillantes recherchent continuellement des failles à exploiter dans ces systèmes informatiques. L'exploitation de failles peut aller de l'installation de virus, au vol de documents confidentiels, aux fraudes ou même à des attaques envers d'autres systèmes. Un problème connu est l'installation de robots sur ces systèmes : des entités malveillantes les installent dans le but de se constituer des armées pour ensuite les utiliser à grande échelle. De telles armées sont en mesure par exemple de faire tomber des systèmes informatiques ou encore d'envoyer des pourriels en grande quantité.

La recherche dans ce domaine est très importante pour la société, beaucoup de personnes reçoivent une quantité considérable de pourriels provenant de ces armées : plus de 70 % [43] des courriels sont des pourriels. Ces pourriels sont souvent de la publicité non désirée, des tentatives de fraudes, d'installation de virus ou tout simplement de propagation de robots. En décembre 2011, McAfee a répertorier plus de 3 500 000 infections de botnets à travers le monde pour ce mois [27] et d'après Abu Rajab *et al.* plus de 27 % du trafic Internet provient des botnets [1].

Plusieurs entreprises travaillent à développer des protections sur leurs systèmes contre de telles menaces (antivirus, pare-feu, filtrage de courriels). Une solution supplémentaire à celle de corriger localement le problème est de déterminer qui est l'auteur de telles

attaques. Cette tâche n'est pas évidente à accomplir, dû au fait que les coordonnateurs de ces armées de robots sont souvent bien cachés. Leurs envois de commandes passent par divers serveurs ce qui leur assure une certaine protection. Pour retrouver les auteurs des attaques, il est donc important de continuer à innover en trouvant de nouveaux moyens de détection de botnets. Il ne faut pas oublier que ces entités malveillantes développent de nouvelles techniques de plus en plus élaborées pour éviter d'être découvertes.

1.2 Définition du projet de recherche

Le but du projet est de développer une infrastructure qui pourra permettre de détecter la provenance d'une attaque de botnet en analysant les entrées et sorties sur des réseaux. Pour le réaliser, une plateforme distribuée sur plusieurs réseaux sera développée. Par la suite, en regroupant les informations obtenues sur ces réseaux, le système tentera de retrouver l'auteur d'une attaque de botnet. Ce travail peut facilement être subdivisé en plusieurs tâches. Pour débuter, il y a la conception de la plateforme distribuée, par la suite l'analyse des données collectées et enfin la réalisation de l'algorithme permettant de faire la corrélation entre les données collectées et la provenance d'une attaque.

La collecte de données à analyser est primordiale pour les étapes subséquentes du système. Pour ce faire, une plateforme distribuée sur le réseau est nécessaire. Ce projet sera consacré principalement au développement de cette plateforme. En général, la quantité de données brutes recueillies est très importante. Par exemple, à l'Université de Sherbrooke, le STI amasse environ 20 Go de données par jour, information qui ne comprend que les adresses de départ et de destination, la durée de connexion et le temps auquel s'est produit ce transfert de paquet. En raison de cette grande quantité de données, un premier filtre devra être développé dans le but de les épurer. Le système complet sera doté de mécanismes pour l'analyse de données. Le présent travail de recherche se limitera cependant à l'élaboration de la partie distribuée et de filtrage initial.

Afin de développer une infrastructure qui pourra permettre la provenance d'une attaque de botnet, le travail devra explorer divers domaines de l'informatique tels que : le traitement de données, la collaboration inter-système et le traitement distribué de données.

1.3 Objectifs du projet de recherche

L'objectif principal de ce projet est le développement d'une plateforme distribuée qui sera en mesure de rassembler des données issues de divers réseaux et d'en extraire l'information essentielle. Cet objectif peut être divisé en plusieurs sous-objectifs :

- Concevoir une plateforme distribuée à l'aide d'outils existants spécifiques au problème des botnets ;
- Concevoir une architecture collaborative ;
- Concevoir un module pouvant prétraiter au minimum 20 Go par jour en vue d'identifier les communications suspectes ;
- Concevoir un module de persistance de l'information ;
- Tester le système à très petite échelle sur un réseau de quelques machines virtuelles ;
- Valider le système développé avec des données provenant du Service des Technologies de l'information de l'Université de Sherbrooke.

1.4 Contributions originales

D'un point de vue global, les travaux de recherches apporteront un nouvel outil pour la sécurité informatique. Celui-ci permettra d'identifier la source d'une attaque de botnet. La contribution apportée par cette phase du projet sera le développement d'un système distribué collaboratif, ce qui permettra d'acquérir un maximum de données pertinentes pour retracer la provenance d'une attaque et de mettre en place des mécanismes de collaboration entre des modules dans plusieurs réseaux.

Actuellement, la détection de botnets est réalisée à l'aide d'outils qui analysent les données d'un réseau spécifique. Alors, concevoir une plateforme permettant de partager ces données en temps réel sur différents réseaux et de collaborer sera un progrès dans le domaine de la sécurité informatique.

1.5 Plan du document

Ce document est divisé en cinq chapitres : l'état de l'art, le développement, une généralisation de la trousse de développement logiciel et l'analyse des résultats.

Quatre sujets sont abordés dans le chapitre de l'état de l'art : une mise en contexte sur la problématique des botnets, les moyens existants pour détecter un botnet, une présentation de systèmes distribués existants et la présentation d'outils qui pourraient s'avérer utiles pour l'accomplissement de ce projet.

Le chapitre suivant qui présente le développement réalisé lors de ce projet est divisé en trois sections : la première présente l'architecture globale de la plateforme distribuée collaborative, la deuxième présente une trousse de développement logiciel et pour finir, la dernière section traite de l'implémentation des modules de la plateforme distribuée collaborative.

Le chapitre de généralisation de la trousse de développement explique comment le SDK implémenté lors de ce projet pourra être réutilisé dans l'implémentation de systèmes distribués et quels seraient les avantages de l'utiliser. Ce chapitre démontre par des exemples simples les possibilités qu'offre cette trousse de développement logiciel.

Le chapitre suivant présente l'architecture de tests utilisée, les résultats des tests ainsi que l'analyse de ceux-ci.

Pour compléter ce mémoire, le dernier chapitre offre une conclusion à l'ensemble, résumant le travail accompli ainsi que les travaux futurs qui pourraient suivre.

CHAPITRE 2

ÉTAT DE L'ART

Ce travail de recherche a pour objectif de produire une infrastructure de calcul distribué destinée à la détection de certains logiciels malveillants. En particulier, les logiciels malveillants fonctionnant en réseau, comme les botnets, sont visés par le système proposé. Ce chapitre est séparé en quatre parties : la première explique ce qu'est un botnet, comment il fonctionne, et en quoi il représente un problème important. La deuxième partie présente un ensemble d'outils existants, destinés spécifiquement à la détection de botnets. La troisième partie passe en revue un ensemble de techniques et systèmes existants permettant d'effectuer des calculs distribués de façon efficace. Cette troisième partie touche directement ce qui a été réalisé dans le cadre de ce travail de recherche : une infrastructure hiérarchique de calcul distribué permettant le traitement ainsi que l'échange d'information entre systèmes distants. Finalement, la quatrième partie présente des outils qui pourraient s'avérer utiles pour l'accomplissement de ce projet.

2.1 Cadre de référence

Dans le but de comprendre les motifs de la recherche, il est important d'expliquer ce qu'est un botnet : c'est un regroupement d'ordinateurs infectés par un logiciel malveillant, où chaque ordinateur est appelé un robot. Le logiciel malveillant permet aux coordonnateurs de ces armées de prendre le contrôle de l'ordinateur infecté par diverses techniques et souvent cela se fait dans le but de commettre des actes répréhensibles. Ce qui rend difficile la détection de botnets, et surtout l'identification du serveur qui les commande et les contrôle, c'est qu'ils constituent des regroupements distribués sur les ordinateurs du monde entier, et que les communications entre les robots et leur serveur de commande sont souvent très furtives et indirectes. Cependant, les «patterns» (motifs) d'interaction entre ceux-ci étant similaires, un système distribué à grande échelle aurait plus de chance de les identifier.

Pour être en mesure de détecter un botnet, il est bon de connaître comment il fonctionne. Le modèle de Kok et Kurz [23] décrit le cycle de vie d'un botnet en cinq étapes :

- La création du robot (écriture du logiciel malveillant) ;
- L'infection d'un ordinateur (installation du logiciel sur un ordinateur) ;
- Union avec le serveur de commande et contrôle ;
- Attentes des commandes ;
- Exécution des commandes.

Pour ce qui est de la phase d'infection d'un ordinateur, plusieurs moyens sont utilisés tels que : l'utilisation de courriels malicieux, l'utilisation des vulnérabilités d'un logiciel ou d'un système d'exploitation, les messages instantanés, les réseaux d'échange de fichiers ou encore par l'entremise d'un autre botnet [17].

Par la suite, le robot doit établir une communication avec son maître. Pour ce faire, plusieurs méthodes existent. Différents protocoles de communication sont utilisés dans le but de se camoufler dans les communications présentes sur le réseau et du fait même éviter les systèmes de détection d'intrusion. Parmi les protocoles fréquemment utilisés pour communiquer avec le serveur de commande et contrôle, deux sont très répandus soit IRC et HTTP [12]. De plus, d'autres moyens existent pour aider les robots à se camoufler tel que les réseaux Fast-Flux [55]. Dans le but de décentraliser les communications et les rendre plus résistantes aux fermetures de serveur, des architectures de botnets poste-à-poste (P2P) ont été développées [17].

Une des difficultés pour retracer les communications du coordonnateur avec son botnet est l'utilisation de protocoles de communication rendant le coordonnateur anonyme. L'un des protocoles fréquemment utilisé est Tor [45]. Son utilisation permet de transiger de l'information sur Internet de façon anonyme. Pour ce faire, un réseau a été mis en place et il permet aux individus qui l'utilisent d'avoir accès à des points de sorties pour leur communication dans plusieurs pays. En plus d'avoir des points de sortie partout dans le monde, les communications entre ces points sont chiffrées avec des clés AES-128 [56]. Il est légitime d'utiliser le protocole Tor pour se protéger sur Internet, cependant il peut aussi être utilisé pour commettre des actes criminels tout en permettant à l'auteur de ces actes criminels de rester anonyme [10].

Une fois la communication établie, le robot peut passer à la phase exécution des commandes. Les principales actions demandées aux botnets d'après Symantec Cloud Message Labs Intelligence [42] sont :

- Envoi de pourriels en grande quantité ;
- Attaque de destruction (Déni de service, déni de service distribué) ;
- Propagation du botnet.

Dû à la menace constante des botnets sur Internet, des groupes de recherche ont entrepris de développer de nouveaux systèmes de détection de botnets. Leurs recherches se spécialisent sur différentes phases du cycle de vie d'un botnet. Les phases les plus explorées sont l'infection et l'exécution de commandes. Pour ces phases, des outils et prototypes existants seront présentés dans les prochaines sections. Dans la section concernant l'analyse des outils pour la phase d'infection, les sujets explorés seront le projet Honeynets [44] et les systèmes de détection d'intrusion et d'anomalie. Pour ce qui est de la section analyse des outils pour la phase d'exécution, des systèmes complets seront explorés tels que : BotHunter [8], BotMiner [16] et BotSniffer [15], en plus de prototypes de détection basés sur une méthode spécifique. Une section sera dédiée aux modèles d'architecture pour la détection de botnet. Finalement, un outil permettant d'analyser les communications d'un botnet sera présenté (Masterblaster [34]).

2.2 Analyse des outils pour la phase d'infection

Dans cette section, une analyse de différents outils sera présentée. Les outils explorés sont ceux qui permettent d'identifier de nouveaux botnets, tels que les Honeynets et des outils qui permettent de détecter qu'une infection est en cours tels que : les outils de détection d'intrusions et d'anomalies, Snort et Ourmon.

2.2.1 Le projet honeynet

Le projet Honeynet est un outil important, qui laissent un logiciel malveillant l'infecter pour ensuite observer son comportement. Ceci permet de comprendre comment l'ordinateur a été infecté et de quelle nature est le logiciel malveillant. Pour être en mesure de capturer des logiciels malveillants, l'outil est un ensemble de pots de miels (honeypots). Les pots de miel sont des ordinateurs plus ou moins vulnérables qui ont pour mission d'être infectés par un logiciel malveillant et par la suite de permettre l'analyse de celui-ci.

Ce projet est en mesure de faire du contrôle, de l'enregistrement et de l'analyse de données [40, 53]. Une fois le système infecté par un logiciel malveillant, il fait une analyse sur son système d'exploitation dans le but de déterminer comment il a été infecté et d'où provient l'infection. Deux techniques sont fréquemment utilisées pour contracter un logiciel malveillant, la faible interaction du client et la haute interaction du client.

La première méthode est un simulateur qui est installé sur un ordinateur et visite des sites web dans le but de faire infecter ce pot de miel, l'analyse des données est faite sur une base de signature permettant ainsi l'identification de façon unique du logiciel malveillant.

La deuxième méthode consiste en une personne qui visite des sites internet dans le but de se faire infecter. Pour cette méthode une analyse plus poussée du système est effectuée après chaque visite de serveurs, ce qui permet de détecter une attaque inconnue [44].

Le Projet Honeynet est un bon outil pour la collecte des données, l'identification de nouveaux logiciels malveillants et l'analyse du comportement de ces logiciels malveillants. En raison de sa nature, il est en mesure de détecter de nouveaux logiciels malveillants qui s'installent sur ses noeuds. Cependant, il n'est pas en mesure de détecter ou de traiter des communications suspectes sur un réseau et de détecter un botnet sur un autre ordinateur.

2.2.2 Les systèmes de détection d'intrusions

Pour réussir à infecter un système informatique, plusieurs méthodes existent, par exemple l'intrusion sur un réseau. Dans ce cas, l'utilisation de vulnérabilités présentes sur le réseau sont utilisées afin d'infecter les différents noeuds. Il existe des outils pour détecter ces types d'attaques. Habituellement, ces outils analysent les communications se propageant sur le réseau pour déterminer s'il y a intrusion ou non. Lorsqu'une intrusion est détectée, une alerte peut être envoyée à l'administrateur pour qu'il puisse agir en conséquence.

Snort [39] est l'un de ces systèmes. Pour détecter une intrusion, il utilise une série de règles. Celles-ci sont fréquemment mises à jour via internet, dans le but d'être en mesure de détecter les nouvelles menaces. Ce système peut être configuré en trois modes soit : renifleur, enregistreur de paquet et détection d'intrusion. Chacune de ces configurations a ses utilités. Par exemple des logiciels peuvent s'installer en parallèle avec Snort et analyser les données collectées par celui-ci. Dans les sections subséquentes, des logiciels de détection de botnets qui utilisent ce principe seront expliqués. Snort intègre plusieurs outils, dont un module de reconnaissance des protocoles et d'autres pour détecter certaines attaques tel qu'une attaque de déni de service [24].

Le principal mandat de Snort est de surveiller un sous-réseau spécifique, ce qui le prive d'information sur la situation globale d'un botnet ou de tout autre type d'attaque.

2.2.3 Les systèmes de détection d'anomalies

Plusieurs types d'anomalies peuvent se produire sur un réseau informatique et c'est pour cette raison que des systèmes ont été conçus pour les détecter. Ces types d'anomalies pouvant être des balayages réseau ou encore des attaques de déni de service.

L'un de ces systèmes est Ourmon [7], qui est basé sur la détection d'anomalies causées par les clients du réseau. En d'autres termes, si un client sur le réseau tente de balayer le réseau ou encore de commettre un déni de service, une alerte sera créée dans Ourmon. Jim Binkley, professeur de l'Université d'État de Portland, a adapté ce système pour détecter la présence d'un botnet. Cette adaptation a été possible parce que les botnets ont souvent ce type de comportement (balayage de réseau, déni de service, envoi de pourriel en grande quantité, etc.). Le système Ourmon est doté d'une interface graphique permettant de visualiser l'état du réseau en continu. Pour déterminer la présence d'une anomalie, des techniques d'exploration de données sont utilisées tel : le minage de données [36].

2.3 Analyse des outils pour la phase d'exécution

Cette section porte sur les outils de détection de botnets lors de la phase d'exécution. Ces outils doivent être en mesure, une fois qu'un ordinateur a été infecté, de déterminer la présence d'un botnet. Des outils sur le marché accomplissent cette tâche telle que : BotHunter [8], BotSniffer [15] et BotMiner [16]. De plus, des prototypes ont été conçus dans le but de tester des méthodes de détection spécifiques.

2.3.1 Les systèmes existants

Bothunter

Bothunter est un système de détection de botnets. Pour faire cette détection, il identifie les tentatives de propagation d'un botnet sur le réseau. Pour atteindre cet objectif, Bothunter utilise un algorithme de corrélation basé sur l'analyse des communications sur le réseau [15, 35]. L'analyse des communications sur le réseau est faite par le système d'identification d'intrusion Snort [8, 39].

Lorsque Bothunter identifie un botnet potentiel, il compare la signature du botnet avec sa base de données. Une fois la signature validée, il est en mesure de communiquer les résultats de façon confidentielle à l'aide du protocole Tor au serveur principal [45]. Le rôle du serveur principal est de recueillir l'information sur les différentes attaques se produisant sur les différents réseaux [8].

La principale caractéristique de Bothunter est la détection de botnets, basée sur les tentatives de propagation du botnet. Il peut aussi être considéré comme étant distribué vu qu'il informe le serveur principal des attaques se produisant sur son réseau. Cependant, il est limité à la détection de botnets qui ont une certaine signature. De plus, il analyse les données sur un réseau spécifique, ce qui l'empêche de voir la situation globale d'une attaque.

Botsniffer

Botsniffer est un outil qui détecte les botnets qui sont contrôlés par un serveur de commandes et contrôle sous les protocoles IRC et HTTP. Pour détecter des botnets sur un réseau, il analyse les similitudes dans les communications réseau sur une fenêtre temporelle.

Pour la réalisation de ce logiciel, l'auteur a supposé que tous les ordinateurs infectés par le même botnet réagissent de façon similaire. Ces comportements similaires permettent de regrouper les ordinateurs infectés et de définir le botnet.

Botsniffer a plusieurs phases de traitement, la première étant de comparer les communications avec une liste blanche statique et la deuxième avec une liste blanche dynamique, qui permettent d'éliminer efficacement les connexions à des sites reconnus comme étant sans danger. Par la suite, les données sont passées dans un analyseur de protocole et un outil de détection d'activité de réponse, ces modules génèrent des journaux d'activités. Par la suite, un module de corrélation analyse ces données pour déterminer la présence ou non d'un botnet. À la suite d'une détection de botnet, l'outil émet une alerte aux administrateurs du réseau.

L'outil Botsniffer est conçu pour être distribué, le premier module, celui d'enregistrement des données, peut être placé à plusieurs endroits sur le réseau. Par la suite, les données sont transférées au module de corrélation qui fait l'analyse de celles-ci [15].

Botsniffer permet de distribuer ses modules sur plusieurs machines dans un réseau. Ce qui permet de faire la collecte des données et l'analyse de celles-ci à des endroits différents. Si le module de collecte des données est distribué sur plusieurs réseaux, les données sont toutes envoyées au même point de traitement. Aussi, une distribution massive du module

de collecte de données pourrait engendrer une surcharge de travail pour le module de corrélation des données.

Botminer

L'outil Botminer a pour but de trouver un groupe d'ordinateurs infectés dans un réseau de façon passive. Il ne doit pas être limité à un type de communication ou un type d'architecture, de plus le fait qu'une communication soit chiffrée ou non ne doit pas avoir d'impact sur son analyse. Pour ce faire, il est construit en cinq modules qui sont décrits plus bas.

Le premier module est nommé le Plan-A, ce module est responsable de détecter quel ordinateur fait quoi dans le réseau. Pour ce faire, Botminer utilise Snort pour extraire des exceptions.

Par la suite, le Plan-C est responsable de déterminer quels ordinateurs parlent avec quels autres ordinateurs et de réduire ce trafic à une taille acceptable.

Ces deux premières étapes se font en parallèle, par la suite, la sortie des plans est passée dans un module de regroupement des données.

L'auteur de Botminer a indiqué que la prochaine étape du développement de son outil devrait inclure un aspect de distribution sur plusieurs réseaux avec un serveur qui gèrerait le tout, ce qui apporterait une vue plus globale du problème. De plus, il discute des problèmes que son système pourrait rencontrer. Il dit que si un botnet utilise des sites reconnus comme sécuritaires provenant des listes blanches (Google, Yahoo), alors le botnet pourrait réussir à se camoufler [15, 16].

2.3.2 Algorithmes variés

Détection basée sur l'analyse fréquentielle

AsSadhan *et al.* [6] présente un modèle de détection de botnets basé sur l'analyse fréquentielle des communications entre un botnet et un serveur de commande et contrôle. Cet outil recherche la périodicité dans les communications entre l'ordinateur infecté et le serveur auquel il répond en recherchant des adresses IP et des ports de communication semblables. Les résultats démontrent le bon fonctionnement de cette méthode, même sur un réseau bruité par d'autres communications. Cependant, une faille majeure subsiste et l'auteur en est conscient : de plus en plus les botnets communiquent de façon aléatoire avec le serveur de commande et contrôle dans le but de ne pas se faire détecter, ce qui vient déjouer ce système.

Détection de trafic IRC réel ou non

L'un des moyens de communication des botnets avec les serveurs de commandes et contrôle est le protocole IRC. Lin *et al.* [25] présente un modèle pour la détection de trafic IRC anormal. Pour ce faire, l'analyse est faite en deux étapes principales. La première est la détection de présence ou d'absence de trafic IRC. Une fois que le protocole IRC a été identifié, en analysant le flux de données, l'outil est en mesure de déterminer si ce trafic est normal ou non. Ils ont déterminé qu'une communication avec un serveur de commande et contrôle avait souvent un temps de réponse très rapide [25]. Cet outil permet donc de détecter un trafic anormal d'IRC. Malheureusement, il a été observé récemment que les serveurs de commandes et contrôle des botnets migrent vers d'autres types de protocoles de communication tels que HTTP pour l'utilisation de requête web qui se dissimule bien sur un réseau ou une architecture P2P avec des protocoles de communications propriétaire ce qui rend ce modèle plus difficile d'utilisation. De plus, se baser sur le temps de réponse plus ou moins rapide des communications peut facilement être déjoué. Le botnet aurait seulement à introduire des délais aléatoires pour réussir à se camoufler, ou même à se baser sur des statistiques de frappe réelle pour passer inaperçu.

Détection des réseaux poste-à-poste

Liu *et al.* [26] a développé un modèle de détection de botnet P2P. Ce modèle est composé de trois sous-systèmes : la détection de noeuds P2P, le regroupement des noeuds et la détection de botnet.

Pour ce qui est de la détection de noeuds P2P, l'auteur s'est basé sur deux définitions, le lien entre les noeuds et la distribution des noeuds. Pour évaluer la première définition, le lien entre les noeuds, l'algorithme regarde le nombre de connexions présentes sur un noeud vers un autre noeud et lui attribue une valeur qui est relative à la quantité de connexions effectuées entre ceux-ci. Pour ce qui est de la deuxième définition, l'algorithme analyse si les noeuds potentiels se trouvent sur le même réseau ou encore sur le même sous-réseau, l'algorithme attribue alors une valeur basée sur cette information. Ces deux valeurs sont ensuite combinées, si le résultat de ces deux valeurs est supérieur à un seuil fixé par l'auteur alors, l'algorithme a détecté un noeud P2P.

Par la suite, le système tente de regrouper les noeuds. Pour ce faire, trois définitions sont présentes dans le système, soit le degré de connexion entre les noeuds, le degré de symétrie et la distance entre les noeuds. L'auteur se base sur le principe que les noeuds d'un réseau P2P devraient avoir des caractéristiques semblables pour les regrouper. Pour regrouper ces réseaux,un algorithme basé sur la méthode de la K-moyenne est utilisé [11].

Enfin pour la détection de botnet, cinq caractéristiques sont considérées : soit le balayage de réseau, l'envoi de pourriel, le téléchargement, l'exploitation du code et les dénis de service distribué. Un seuil est fixé et si l'observation de ces comportements dépasse ce seuil alors le réseau P2P est défini comme un botnet.

Utilisation de réseaux de neurones

Nogueira *et al.* [29] a développé un système pouvant identifier les types de trafic présent sur le réseau. Pour cela, il utilise un réseau de neurones. Le flux sur le réseau est analysé en premier lieu, des caractéristiques comme la taille des paquets et le nombre de paquets sur un segment de réseau sont extraites. Par la suite, celles-ci sont présentées au réseau de neurones qui détermine quel est le type de trafic sur le réseau. Le système est basé sur le fait que chaque type de communication utilise le réseau d'une certaine façon, caractérisée par un niveau d'activité, une périodicité ou une certaine constance.

2.4 Analyse des outils d'identification des attaquants

Il est important de pouvoir déterminer si un ordinateur est infecté ou non par un botnet. Cependant, pour réussir à entièrement éradiquer le problème, il faut identifier la source. Pour ce faire, il faut retrouver les coordonnateurs des botnets.

MasterBlaster [34] est un outil qui a été développé dans le but d'analyser les botnets et de retrouver de l'information pertinente concernant les coordonnateurs de botnets. Pour ce faire, le système classifie les botnets en trois catégories : à but destructif, à but financier et à but indéterminé. Par la suite, en analysant les communications des botnets, il est en mesure de déterminer le surnom des coordonnateurs de ce botnet. Une fois le surnom des coordonnateurs identifiés, des corrélations sont faites avec la nature du botnet et les attaques qu'il produit. De leurs analyses, ils ont pu déterminer que des coordonnateurs se spécialisaient souvent dans un type d'attaque et que plusieurs coordonnateurs sont présents sur le botnet dans le but d'exploiter les autres facettes de celui-ci.

Il existe quelques limitations à cet outil. Premièrement, il est limité aux botnets de type IRC et deuxièmement, si les communications sont cryptées, il ne peut en faire ressortir l'information.

2.5 Analyse des architectures collaboratives pour la détection de botnet

2.5.1 Une architecture collaborative

Wang et Gong [49] ont présenté un modèle d'architecture distribuée collaborative, qui permettrait de collecter des données sur divers réseaux et avoir une vue plus globale d'un botnet. Leurs recherches ont consisté à développer une architecture de système collaboratif et distribué, basé sur plusieurs types de modèles.

Le premier modèle étudié est une architecture construite en trois niveaux, les agents intelligents, le gestionnaire et l'administrateur. Voir la figure 2.1.

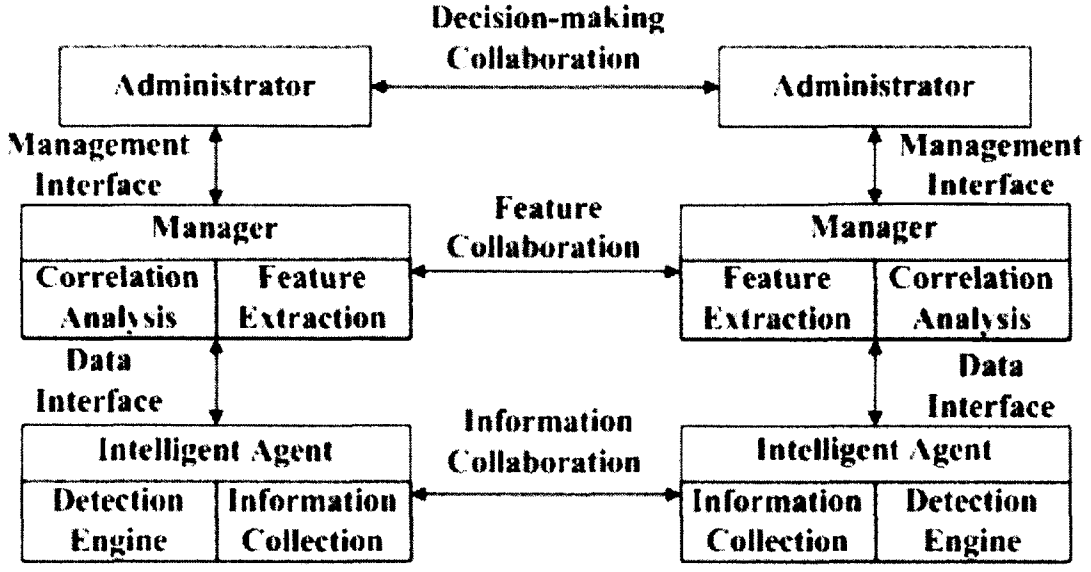

Figure 2.1 Architecture collaborative (Tiré de [49])

Le rôle des agents intelligents est de collecter les données et de faire un premier traitement sur celles-ci à l'aide d'une série de règles. Lorsqu'une alerte est émise par ce niveau, elle est reconduite au deuxième niveau, celui du gestionnaire. Par la suite, celui-ci fait de l'extraction de caractéristiques et fait des corrélations sur les données. Enfin, le rôle de l'administrateur est de rendre une décision concernant l'alerte et statuer si cette dernière était ou non un botnet.

La partie collaborative de ce système est implantée sur chaque niveau de l'architecture. Au niveau des agents intelligents, ils peuvent échanger de l'information avec d'autres agents intelligents. Par la suite, le gestionnaire échange de l'information sur les caractéristiques

avec d'autres gestionnaires et finalement l'administrateur échange de l'information avec les autres administrateurs pour la prise de décision.

2.5.2 Une architecture collaborative basée sur le rôle des agents

Suite à leur premier modèle, Wang et Gong [49] ont présenté un deuxième modèle d'architecture de système distribué collaboratif. Celui-ci est plutôt basé sur des rôles. L'architecture hiérarchique est toujours basée sur un modèle à trois niveaux. Le premier étant le niveau des tâches, le deuxième étant celui de l'organisation des tâches et finalement le troisième étant le niveau de collaboration. Ces trois niveaux sont maintenant regroupés dans un groupe de travail. Voir la figure 2.2.

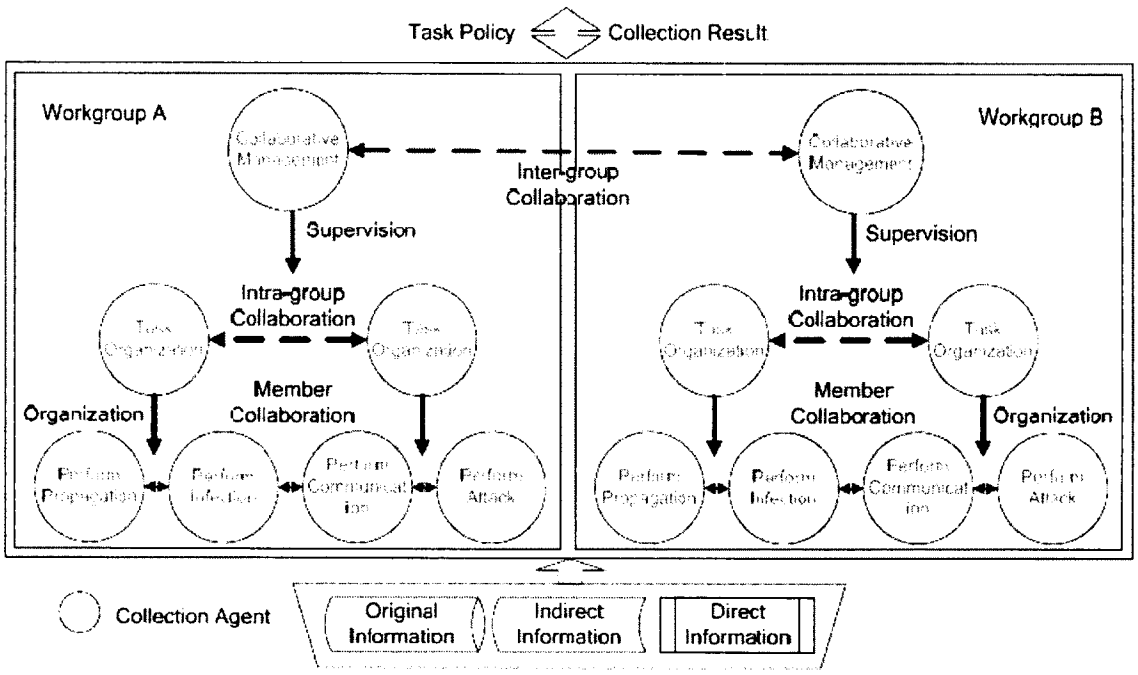

Figure 2.2 Une architecture collaborative basée sur le rôle des agents (Tiré de [51])

Au premier niveau, l'agent peut exécuter quatre types de tâches : une analyse de la propagation, de l'infection, de la communication et des attaques. Ces tâches sont définies de telle sorte qu'il soit plus simple de classifier les données provenant d'un éventuel botnet. Ces tâches sont en mesure de communiquer avec d'autres tâches qui sont dans le même groupe de travail pour échanger de l'information.

Par la suite, le deuxième niveau s'occupe de l'organisation des tâches et finalement le troisième niveau prend en charge la communication entre les groupes de travail [51].

2.5.3 Une architecture collaborative avec fusion de senseurs multiples hétérogènes

Par la suite, Wang et Gong [50] ont développé un nouveau modèle basé sur la fusion de données provenant de senseurs hétérogènes. Le système est séparé en deux branches fonctionnant en parallèle, la première étant l'analyse temporelle et l'autre l'analyse spatiale. Voir la figure 2.3.

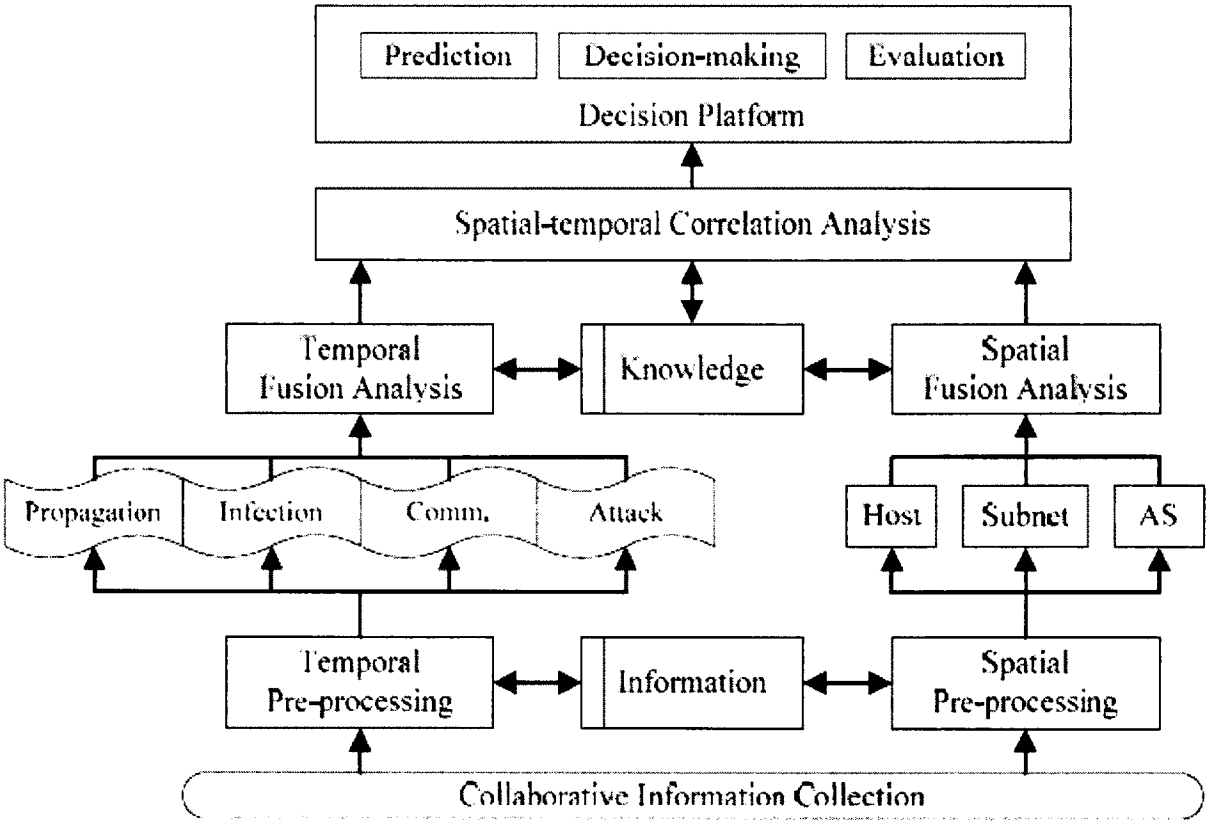

Figure 2.3 Une architecture collaborative avec fusion de senseurs multiples hétérogènes (Tiré de [50])

Pour l'analyse temporelle, la première phase est un traitement temporel de l'information. Ensuite, comme dans leurs travaux précédents [51], ils divisent l'analyse en quatre modules de traitement soit la propagation, l'infection, la communication et l'attaque ce qui représente les phases d'un botnet. Pour finaliser le traitement de cette branche, une fusion des données des quatre modules est faite.

Pour l'analyse spatiale, la branche commence avec un prétraitement, suivi de modules de traitements des données tels que : un client, un sous-réseau, etc. Ces modules représentent des entités physiques. Un module fusionne l'information provenant des modules précédents sur une même branche.

Finalement, un module fusionne les données des deux branches, temporelles et spatiales, pour que le module de décision soit en mesure d'évaluer la situation actuelle du botnet et de la faire parvenir aux administrateurs du réseau [50, 52].

Ce système est encore jeune. Le but de l'auteur est d'implémenter le prototype dans le futur. Présentement, uniquement un modèle sur papier a été pensé et aucune mise en oeuvre n'a été réalisée.

Ce type d'architecture peut être intéressante, car elle sépare les tâches de traitement temporelles des tâches spatiales. Elle est dite collaborative vu qu'elle est en mesure de traiter l'information provenant de divers systèmes tels que : ordinateur, serveur, routeur, etc. Cependant, elle est conçue pour être utilisée de façon locale, sur un sous-réseau spécifique. Donc, elle ne permet pas d'avoir une vue globale sur l'ensemble d'une attaque d'un botnet.

2.6 Résumé comparatif des techniques de détection de botnet

Les différentes techniques de détection de botnet vues dans ce chapitre sont regroupées au tableau 2.1 qui récapitule leurs principales caractéristiques.

Tableau 2.1 Résumé des outils de détection de botnet

	Analyse	Détection	Protection	Faiblesses
Honeynet [40, 44, 53]	X	X		
Snort [24, 39]		X	X	Sous-réseau spécifique
Ourmon [7, 36]		X		
Bothunter [8, 15, 35]		X	X	Règles définies
Analyse fréquentielle [6]		X		Temps de réponse périodique
Trafic IRC [25]		X		Protocole IRC uniquement
Réseaux P2P [26]		X		Architecture P2P
Réseaux de neurones [29]		X		
Botminer [15, 16]		X		
Botsniffer [15]		X		Protocole IRC et HTTP
Masterblaster [34]	X			Protocole IRC non crypté

Les divers outils présentés sont bâtis principalement pour la détection de botnets. Peu d'entre eux travaillent de façon distribuée et collaborative. De plus, il n'y a pas vraiment d'outils qui existent pour trouver les coordonnateurs des botnets. Certains outils sont en mesure de déterminer le serveur de commandes et contrôle (Botsniffer), alors qu'un

seul outil (MasterBlaster) permet d'analyser les communications des botnets utilisant le protocole IRC.

2.7 Systèmes distribués

Puisque le but de ce projet est d'implémenter une plateforme distribuée collaborative, il est important d'étudier les divers systèmes existants. Quelques systèmes en code libre sont disponibles pour accomplir diverses tâches qui peuvent aider à ce projet. Dans cette section, trois systèmes distincts seront présentés, Storm [41], Hadoop [18] et Orocos [33].

2.7.1 Storm

Storm est une plateforme distribuée pour accomplir des traitements en temps réel. Elle est principalement utilisée dans le but de traiter des grandes charges de données. Afin d'arriver à traiter cette charge, les calculs sont distribués sur un groupe de serveurs. Pour ce faire, Storm utilise un gestionnaire maitre (Nimbus) pour déterminer quelle tâche doit être exécutée sur ces serveurs. Par la suite, le gestionnaire de groupe Zookeeper d'Apache [4] est présent afin de gérer chacun des noeuds du groupe. Finalement, des superviseurs de tâches s'assurent de démarrer chacune des tâches nécessaires lorsque le gestionnaire maître en donne la commande. La figure 2.4 présente l'architecture de Storm.

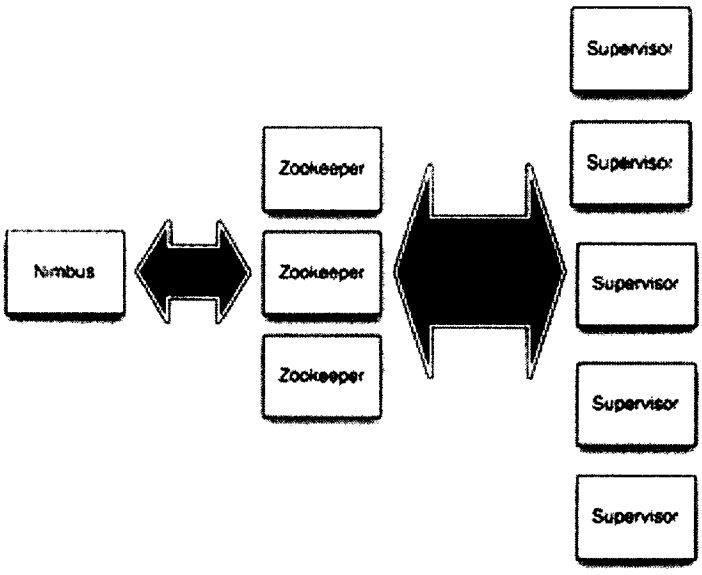

Figure 2.4 Architecture de groupe Storm (tiré de [41])

Une fois l'architecture de Storm installée, le développement peut commencer. Une topologie est composée de : sources d'entrées pour les données (Spout), des tâches (Bolt) et des liens (Stream). La figure 2.5 montre un exemple de topologie.

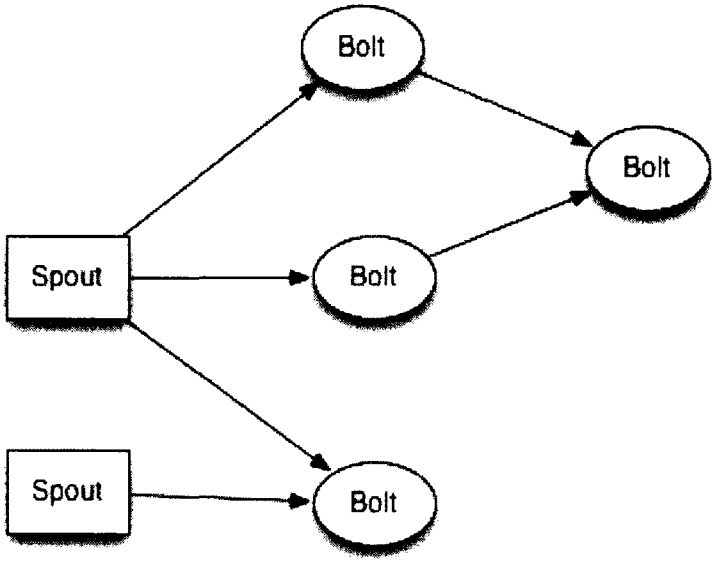

Figure 2.5 Topologie de Storm (tiré de [41])

Le principal attrait de ce système est d'être en mesure de traiter une lourde charge de données à de très grandes vitesses, de s'étendre facilement et de garantir le traitement des données en temps réel.

2.7.2 Hadoop et MapReduce

Le système Hadoop est premièrement un système de fichier distribué libre d'utilisation [38]. Il fonctionne en divisant les fichiers en morceaux qui sont distribués sur plusieurs serveurs et répliqués trois fois pour offrir une redondance. Par exemple Yahoo ! l'utilise, ce qui permet à 25 000 serveurs d'héberger 25 Pétaoctets de données [38]. En plus d'être un système de fichiers distribué, d'autres outils sont intégrés à ce système comme l'algorithme MapReduce [9].

L'algorithme MapReduce permet d'accomplir des calculs distribués en parallèle. Utilisé avec le système Hadoop, il tire profit du fait que les fichiers sont divisés en blocs et distribué sur plusieurs serveurs pour augmenter sa performance. Le but de MapReduce est de fournir un système simple et performant pour accomplir des calculs parallélisés, par exemple des algorithmes de classement des données sont fréquemment implémentés sur

ce système. Cet algorithme s'exécute en deux étapes : le Map et le Reduce. L'opération Map consiste à choisir quelles données seront envoyées à quel travailleur pour accomplir le Reduce d'après une clé définie. Une fois les données placées dans des listes, ces listes sont sauvegardées sur le disque. Enfin, le travailleur qui accomplit le Reduce est averti que des données sont disponibles, les lit, applique l'opération demandée et fournit une réponse dans un fichier (figure 2.6).

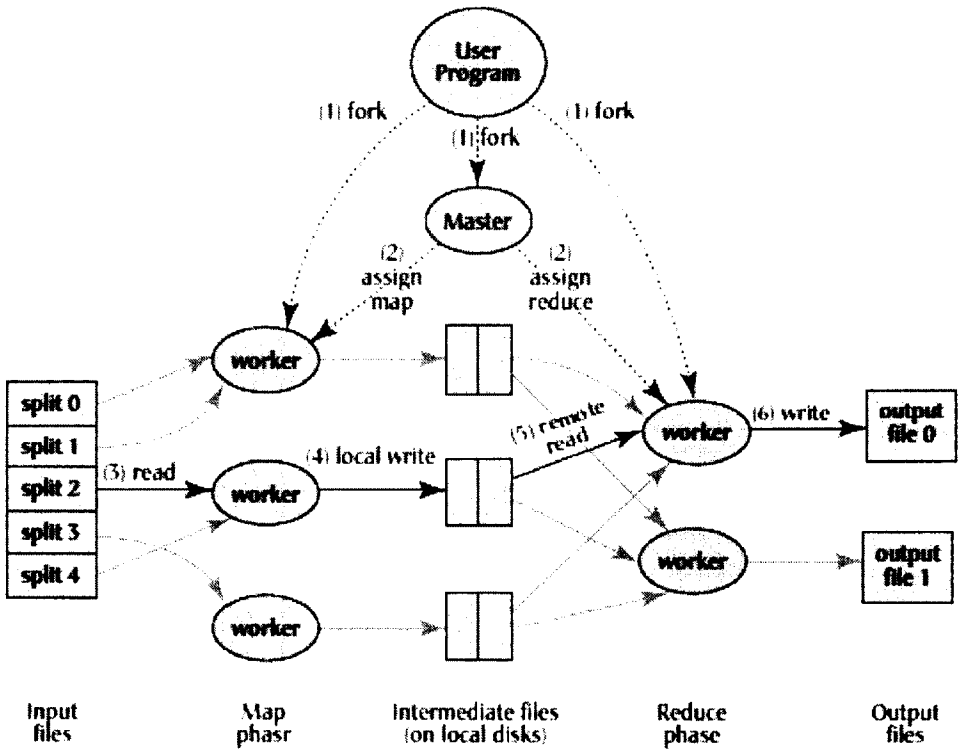

Figure 2.6 Opération de MapReduce (tiré de [9])

2.7.3 Orocos

Orocos [33] est l'acronyme Open Robot Control Software. Cet outil de développement est majoritairement utilisé lors du développement de robot, principalement pour y implémenter des modules. La librairie Orocos est discutée ici en lien avec l'une des options qu'elle offre, la reconfiguration de modules en temps réel. Pour ajouter ces fonctionnalités, l'utilisation de Orocos RTT est nécessaire.

Pour ce faire, lors de la création d'un module, des noms de propriétés sont enregistrés auprès de la librairie. Par la suite, lors de l'exécution à l'aide de méthode «Get» et «Set»

ces valeurs peuvent être changées. De plus, les valeurs par défaut peuvent être placées dans des fichiers de configuration XML afin d'initialiser le module.

L'intérêt du système Orocos dans le contexte de botnets est que ce type de technologie pourrait permettre à un système de s'adapter dynamiquement à une situation d'infection en évolution.

2.8 Courtier de messages

Lors de la conception d'un système distribué, des mécanismes de collaboration entre les divers modules doivent être mis en place. L'une des méthodes possibles pour faire l'échange de messages est l'utilisation de courtiers. Ces derniers offrent plusieurs bénéfices aux applications distribuées qui en font usage. L'un de ces bénéfices importants est de permettre une communication entre plusieurs modules d'un système distribué sans que ceux-ci ne se connaissent. De cette façon, les dépendances entre les divers modules d'un système sont réduites.

Plusieurs types de courtier existent tels que : les courtiers intégrés au logiciel développé, les courtiers autonomes et les courtiers intégrés à des serveurs d'application. Outre le mode d'installation choisi pour un courtier, divers protocoles de communication existent. Les sections suivantes présentent divers types de courtiers avec leurs avantages et inconvénients, ainsi que différents types de protocoles utilisés pour communiquer avec un courtier.

Courtier intégré au logiciel

Puisqu'ils sont intégrés dans le logiciel développé, ils ne nécessitent pas d'installation supplémentaire pour que l'application puisse fonctionner. Leur utilisation en est donc simplifiée. Ces courtiers se caractérisent par leur légèreté et leur facilité à être transporté avec le système.

Cependant, intégrer un courtier à un logiciel peut apporter quelques désavantages. L'intégration du courtier à un logiciel ajoute des dépendances à celui-ci. Il devient beaucoup plus complexe de changer de courtier une fois l'application déployée. De plus, assurer que le courtier est continuellement fonctionnel (High availability) devient aussi un enjeu puisqu'il dépend du logiciel dans lequel il est intégré. Finalement, offrir des services de persistances automatiques des messages s'avère plus complexe.

Un courtier offrant cette possibilité est ActiveMQ d'Apache [2] voir le tableau 2.2 pour la comparaison avec les autres courtiers.

Courtier autonome

Les courtiers autonomes s'installent en tant que service sur un système d'exploitation. Ces courtiers sont donc dépendants des systèmes d'exploitation. Habituellement, ces courtiers se caractérisent par leur légèreté et offrent de multiples possibilités de configurations. Les configurations souvent présentes sont la possibilité d'avoir un service toujours disponible, la possibilité de gérer et de conserver les messages et la possibilité de joindre un groupe de serveur pour distribuer la charge de travail (cluster).

Afin de communiquer avec ce type de courtier, divers types de requête peuvent être utilisés pour réduire les dépendances entre l'application développée et le courtier de sorte qu'il soit possible de changer de courtier de façon transparente pour l'application.

Quelques courtiers offrent ce type d'installation tels que : ActiveMQ de Apache [2], Qpid de Apache [5], RabbitMQ de VMware [46]. Voir le tableau 2.2 pour la comparaison de ceux-ci.

Courtier intégré à un serveur d'application

Afin d'utiliser un courtier de message, il est possible d'installer un serveur d'application tel que : Glassfish [14] ou WebSphere de IBM [54] , deux systèmes qui en fournissent un. Ce type de courtier est comparable à un courtier autonome. Cependant, avoir un serveur d'application ajoute différents services entourant le courtier. Alors, cette installation est plus lourde qu'un courtier autonome.

Standards de communication avec les courtiers

Plusieurs standards existent pour la communication entre les applications et le courtier. De nombreux courtiers supportent Java Message Service (JMS). JMS est intégré au langage Java, alors pour le développement en ce langage ce service est une option intéressante. Cependant, JMS n'offre pas de cryptage de donnée et de mécanisme d'authentification, ceux-ci doivent être fournis par le courtier.

Il existe un deuxième protocole, Advanced Message Queue Protocol (AMQP) [3], qui est aussi d'intérêt. Ce protocole n'est pas dépendant d'un langage. Plusieurs bibliothèques existent pour divers langages de développement. Ce protocole n'est donc pas contraint à un langage de développement. AMQP a été développé afin d'offrir de grandes performances.

Pour finir, d'autres types de protocole propriétaire existent. Par exemple, le courtier Kafka d'Apache [22] implémente son protocole sur TCP/IP.

Tableau 2.2 Comparaison des courtiers de messages

	Type	Caractéristiques
ActiveMQ [2]	Intégré ou autonome	JMS Openwire AMQP
QPID	Autonome	JMS AMQP
RabbitMQ	Autonome	AMQP
Glassfish	Serveur application	JMS HTTP
WebSphere	Serveur application	JMS HTTP
Kafka	Autonome	Propriétaire

CHAPITRE 3

DÉVELOPPEMENT

Ce chapitre présente les accomplissements réalisés lors de ce projet. Puisque l'implémentation de la plateforme distribuée collaborative a été réalisée en plusieurs étapes, trois thèmes sont abordés : la présentation de l'architecture globale, l'implémentation de la trousse de développement logiciel et l'implémentation de la plateforme.

3.1 Architecture distribuée collaborative proposée

Tel que présenté précédemment, les objectifs du projet sont :

- Traiter une grande charge de données ;
- Conserver les données importantes ;
- Faciliter la collaboration entre les divers modules ;
- Être facilement distribuable ;
- Faciliter le développement de nouveaux éléments.

Afin d'atteindre ces objectifs, une architecture distribuée collaborative est présentée (figure 3.1). Cette figure représente un sous-système de la plateforme. Une légende de trois couleurs est utilisée sur le schéma : les modules en bleu ont été créés pour ce projet de recherche, ceux en vert ont été identifiés et choisis parmi des éléments existants, créés par d'autres et finalement ceux en rose sortent du cadre de ce projet.

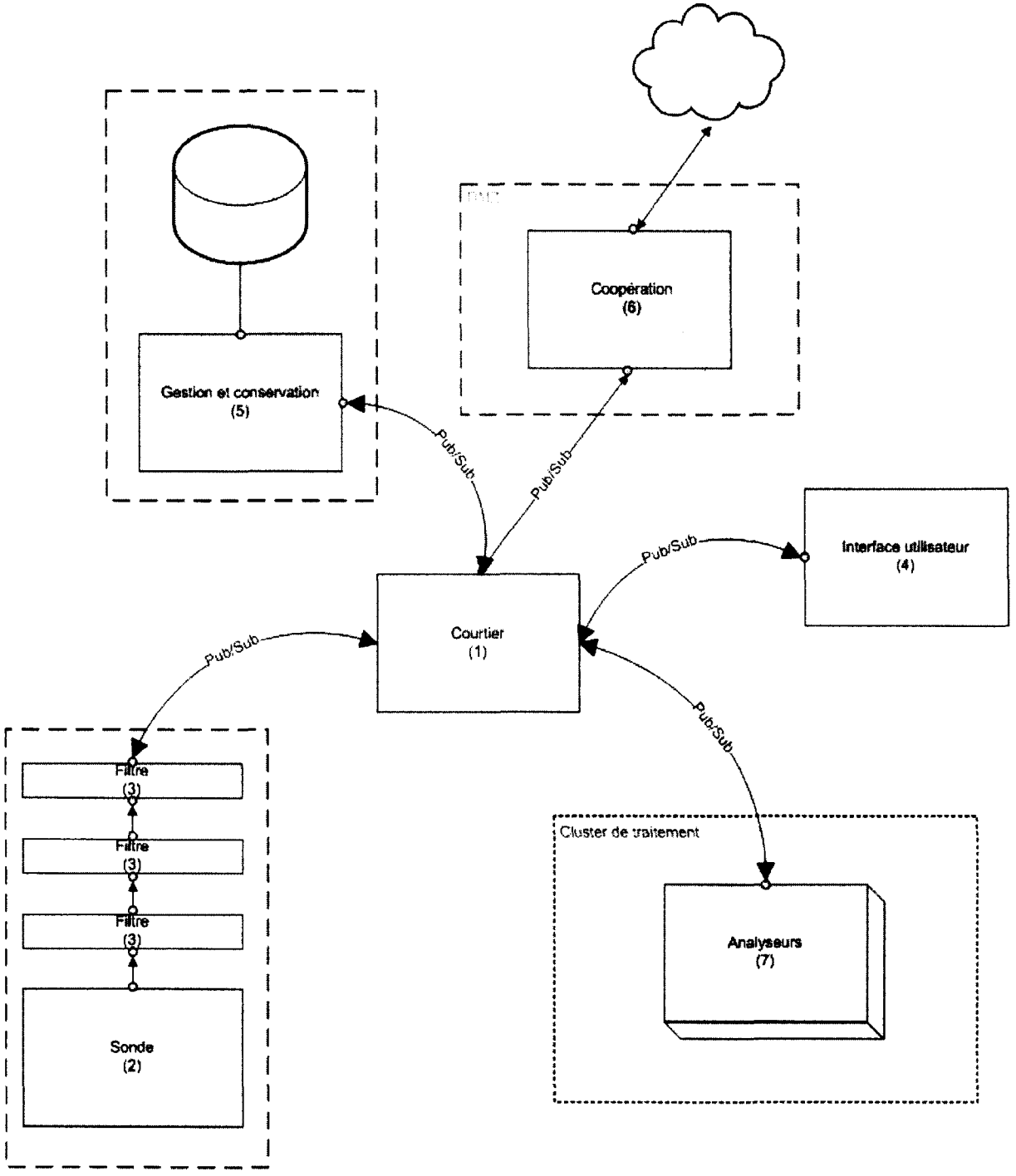

Figure 3.1 Architecture distribuée collaborative proposée

Afin de faciliter le développement et la maintenance futurs du système, les fonctionnalités de la plateforme sont divisées en divers modules. Ces fonctionnalités sont : la lecture de fichiers de journalisation, le filtrage des données, la possibilité de gérer et conserver des données importantes, la configuration de la plateforme en temps réel, la collaboration avec des systèmes distants et l'analyse de données. En conséquence, sept modules sont proposés dans cette architecture pour implémenter ces fonctionnalités :

1. Un courtier ;
2. Un module sonde ;
3. Un module de filtrage ;
4. Un module de configuration ;
5. Un module de gestion et conservation des données ;
6. Un module de collaboration ;
7. Un module d'analyse.

En isolant chaque fonctionnalité dans un module, le système devient plus agile. Chaque module peut être distribué à un (ou plusieurs) endroit précis sur le réseau. Un courtier de messages est intégré au système, pour permettre la communication entre les modules ainsi que leur configuration à distance. Ce courtier est le point central des communications. Les avantages de l'utilisation d'un courtier et le choix de ce courtier sont présentés dans la section suivante.

Le choix de diviser le système en plusieurs modules a été pris dans le but d'apporter des avantages au système tels : traiter une grande charge de données, conserver les données importantes, faciliter la collaboration entre les divers modules, être facilement distribuable et faciliter le développement de nouveaux éléments. En divisant chaque fonctionnalité dans des modules distincts, le système devient plus facile à maintenir, car modifier un module se fera de façon transparente pour le système. De plus, cela permet au système de bien évoluer dans le temps puisqu'il est possible d'ajouter de nouveaux modules offrant de nouvelles fonctionnalités, ce qui permettra de maintenir le système à jour. Enfin, cela fait en sorte que le système soit plus portable et flexible parce que chaque module peut être placé à un endroit stratégique afin d'accomplir son rôle dans le système. Pour finir, puisque chaque tâche est divisée dans un module voir le tableau 3.1, si plus de puissance de calcul est nécessaire, il est possible de démarrer une multitude d'instances d'un module afin de paralléliser le travail à accomplir.

Tableau 3.1 Modules associés avec leurs tâches

Modules	Tâches
Courtier	Gérer les communications dans le système
Sonde	Collecter les données
Filtre	Réduire le nombre de données
Interface utilisateur	Configurer le système
Gestion et conservation	Enregistrer les données
Collaboration	Collaborer avec d'autres systèmes
Analyseurs	Détecter la présence de botnets

Ensemble, les six modules regroupés dans un sous-système donné ont pour objectif d'analyser les communications d'un sous-réseau. Par l'entremise du module de collaboration, chaque sous-système peut communiquer avec d'autres sous-systèmes pour récupérer ou fournir des données nécessaires aux analyses de détection d'attaques logicielles.

Dans le but d'acquérir et de traiter un maximum de données pertinentes à la détection de botnet, ces sous-systèmes devront être installés dans un maximum de réseau tels que : entreprise, université, etc. Une fois cette opération accomplie, le module de collaboration de ces sous-systèmes devra se connecter à un point central tel que présenté à la figure 3.2. Le rôle de la centrale est de mettre en commun les données recueillies dans le sous-système et de les rendre disponibles de façon unifiée aux autres sous-systèmes installés sur le réseau.

Puisqu'une grande quantité de données doivent être traitées afin de retrouver la provenance d'une attaque de botnet, l'analyse devra se faire sur plusieurs niveaux. Pour ce faire, la flexibilité de la plateforme permet de construire un système de traitement hiérarchique. De cette façon, le traitement peut se faire sur plusieurs niveaux (par exemple, une entreprise, une édifice). La plateforme peut être installée dans chacun des édifices et par la suite au niveau du routeur principal de l'entreprise où les données seront regroupées. Lorsque les données sortent de l'entreprise, une plateforme centrale peut être installée par région et finalement par pays. La figure 3.3 démontre les possibilités que peut offrir la plateforme.

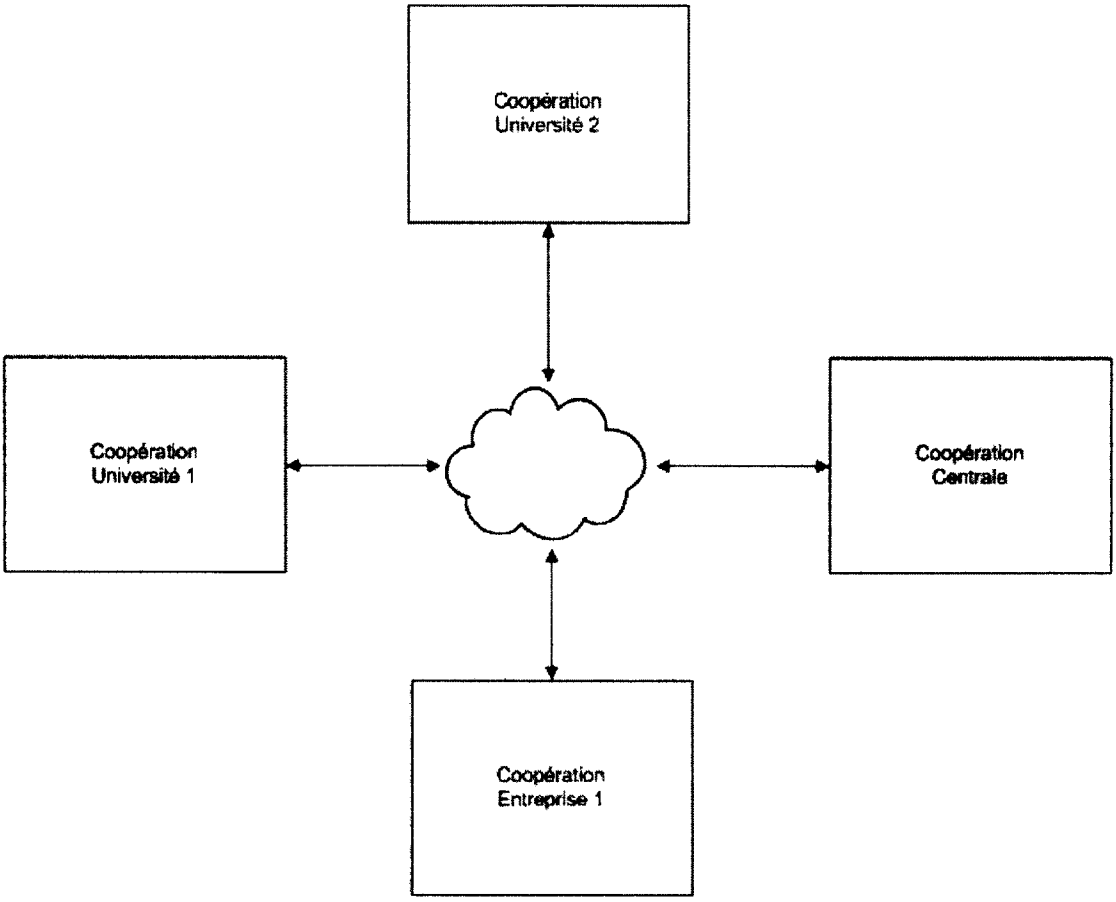

Figure 3.2 Architecture de communication avec l'unité centrale

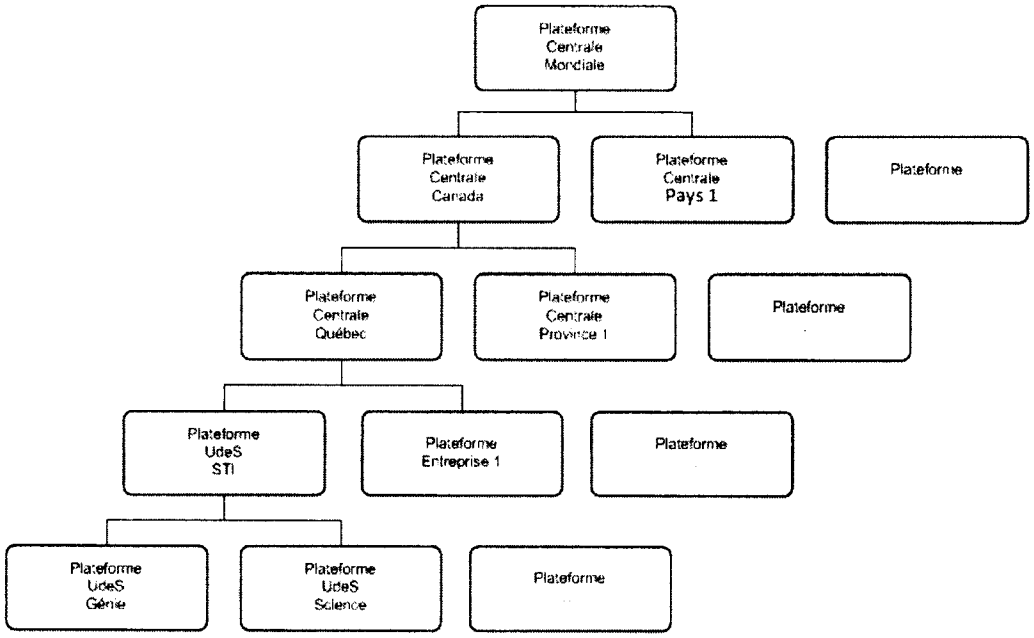

Figure 3.3 Architecture de communication hiérarchique

3.2 La trousse de développement logiciel (SDK)

Cette section traite de la trousse de développement logiciel, de son architecture et des avantages qu'elle offre aux développeurs qui l'utiliseront.

Cette trousse de développement logiciel (SDK) apporte plusieurs avantages lors du développement. Tout d'abord, elle permet de standardiser l'implémentation des modules reliés au système. De plus, le SDK permet de faciliter la maintenance du système de la plateforme, puisque tous les modules sont conçus sous la même architecture et utilisent cette bibliothèque. Enfin, le SDK facilite l'implémentation de nouveaux modules en fournissant une base d'outils génériques.

Afin de standardiser l'implémentation de chaque module, le SDK propose une architecture à respecter lors du développement. Celle-ci est présentée à la figure 3.4. Chaque module est divisé en quatre couches : la couche d'accès à l'information, la couche de métier logique, la couche de traitement et la couche de présentation. Ces couches sont définies dans le SDK par des interfaces et des classes abstraites qui doivent être implémentées afin de créer les fonctionnalités nécessaires à un module. L'explication de chacune des couches est présentée dans les sections suivantes. De plus, un diagramme de classes de haut niveau est présenté à l'annexe A.

Les couches sont implémentées selon le patron de conception observateur [13]. Ceci permet de réduire les dépendances entre chacune des couches et donc de faciliter la maintenance des modules. De plus, en ajoutant cette abstraction entre chaque fonctionnalité dans un module, la modification d'une couche se fait de façon transparente pour le reste des couches de ce module.

Un autre avantage apporté par l'utilisation du SDK est de fournir un ensemble d'outils comme des mécanismes de configurations des modules, des mécanismes de communication, des mécanismes permettant de convertir les messages en objet reconnu par la plateforme et un gestionnaire de données. Ces outils permettent d'accomplir des tâches courantes pour chaque module.

L'implémentation du SDK est en Java [20], ce qui a l'avantage d'en faciliter la portabilité. En développant en Java, les modules peuvent être installés sur divers systèmes d'exploitation pourvu qu'une machine virtuelle Java soit présente sur ces systèmes. De plus, Java utilise la technologie "Just-In-Time" (JIT), ce qui permet d'obtenir un niveau de performance adéquat pour le projet. Le chapitre sur les tests démontrera que la performance du système est tout à fait acceptable.

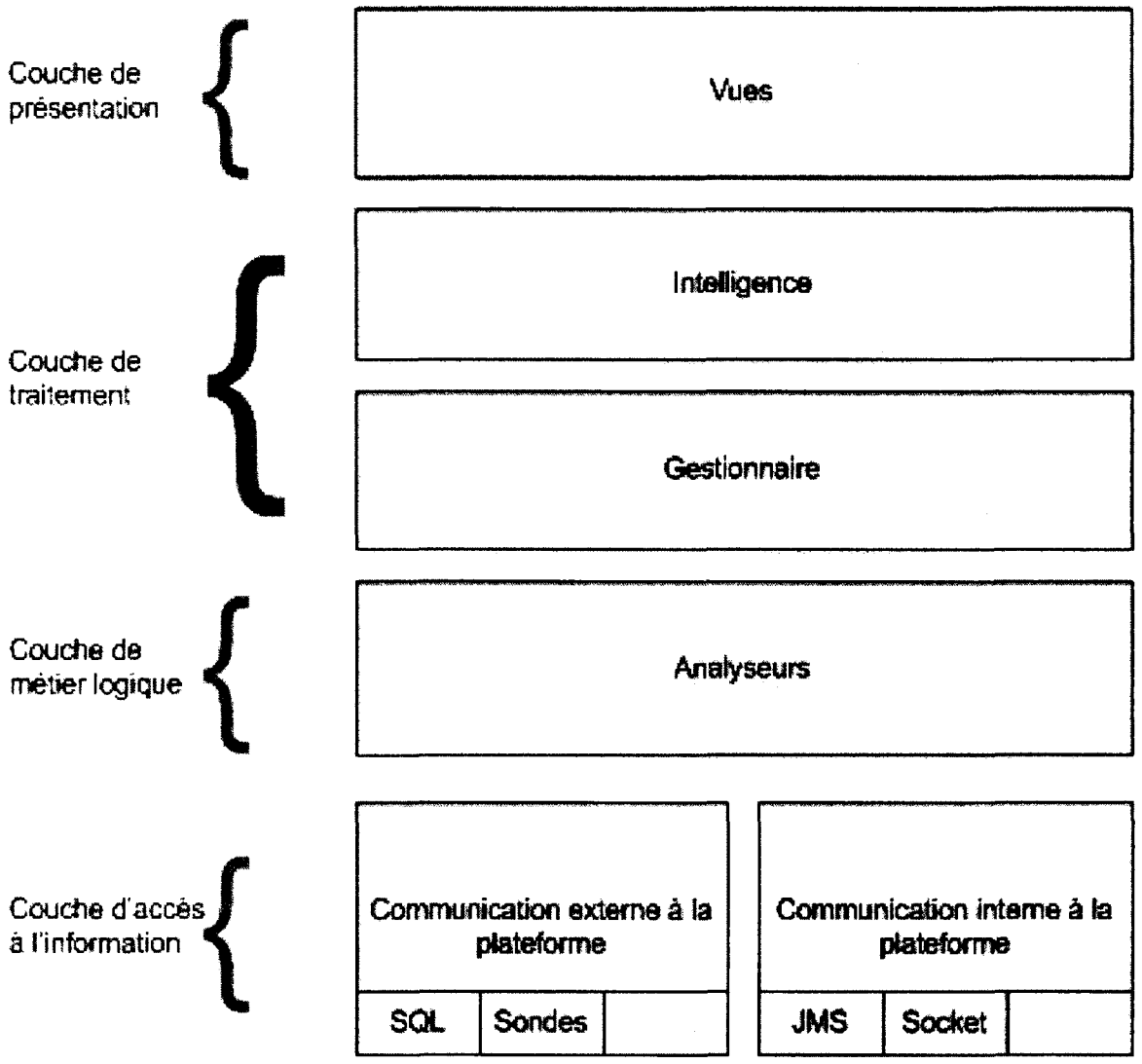

Figure 3.4 Architecture proposée pour un module de la plateforme

Les sections suivantes décrivent le rôle des quatre couches présentes dans la trousse de développement, les outils implémentés associés à celles-ci, ainsi que le développement nécessaire à la création d'un nouveau module.

3.2.1 Couche d'accès à l'information

La couche d'accès à l'information a pour rôle d'assurer les communications avec les autres modules de la plateforme ou avec des systèmes externes. Une abstraction de communication est fournie au niveau de cette couche. Pour ajouter de nouvelles méthodes de communications, cette abstraction doit être implémentée. De cette façon, les couches su-

périeures ne sont pas affectées par le type de communication choisie et cette couche devient parfaitement transparente pour les niveaux supérieurs.

Le patron de conception choisi pour cette couche est l'observateur. L'analyseur de la couche de métier logique s'enregistre auprès de cette couche lors de son initialisation et est averti lorsque de nouvelles données sont prêtes à être traitées.

À ce niveau du module, les données proviennent soit de l'intérieur de la plateforme ou encore de systèmes extérieurs à la plateforme. En d'autres termes, les données provenant de la plateforme sont les communications entre modules et les données provenant de l'extérieur du système sont des données provenant d'entrepôt de données, de fichier de journalisation, etc.

Les mécanismes de communication internes traitent deux types de données, des messages sérialisés et des objets sérialisables. Les données reçues de la couche supérieure en vue d'être transmises doivent être de type objets sérialisables, afin de les convertir en messages. Les messages reçus sont convertis en objets sérialisables. La figure 3.5 montre un exemple de transfert de message entre deux modules.

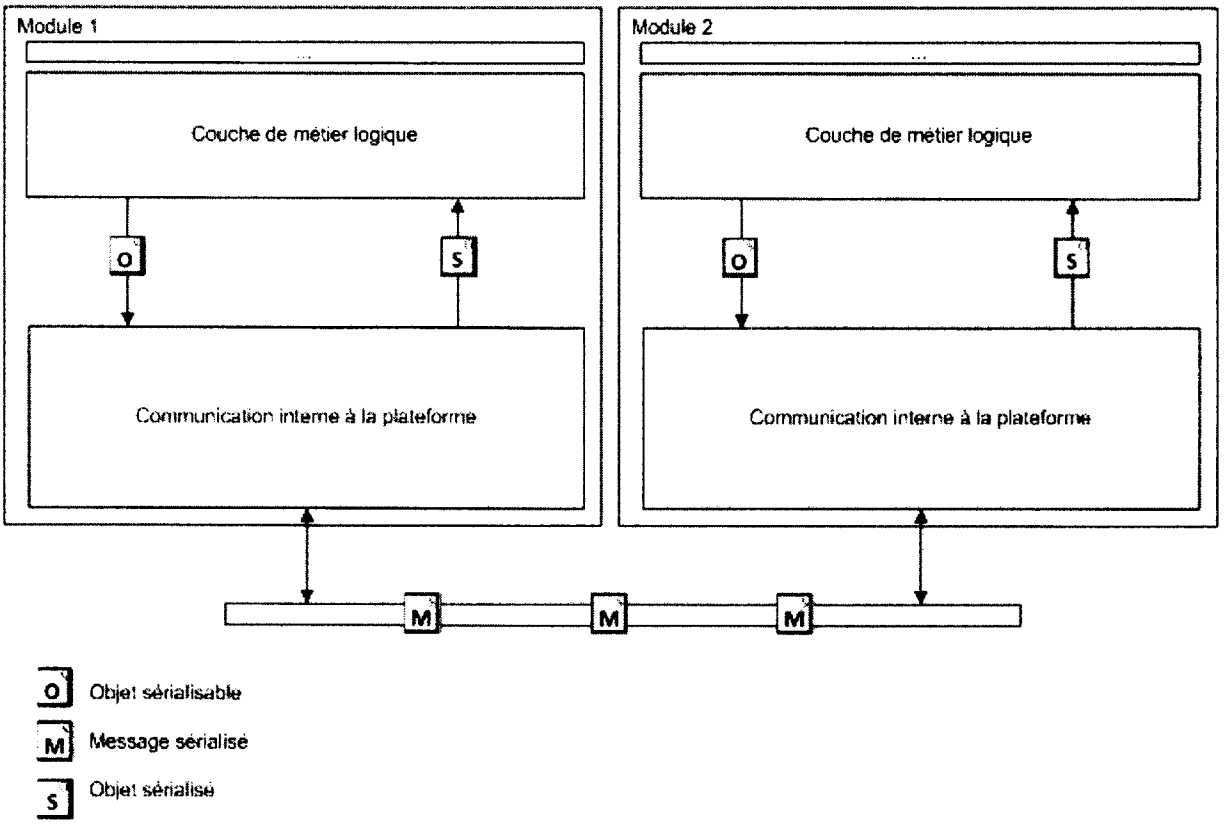

Figure 3.5 Exemple de communication interne au système

Les mécanismes de communication externe traitent deux types de données : des commandes et des messages. Les messages provenant de l'extérieure du système peuvent être des données d'un entrepôt de données (base de données SQL), d'une sonde, etc. Les commandes sont des requêtes pour le mécanisme, soit de type sélection de données ou de persistance de données. La figure 3.6 expose une communication externe avec une base de données SQL.

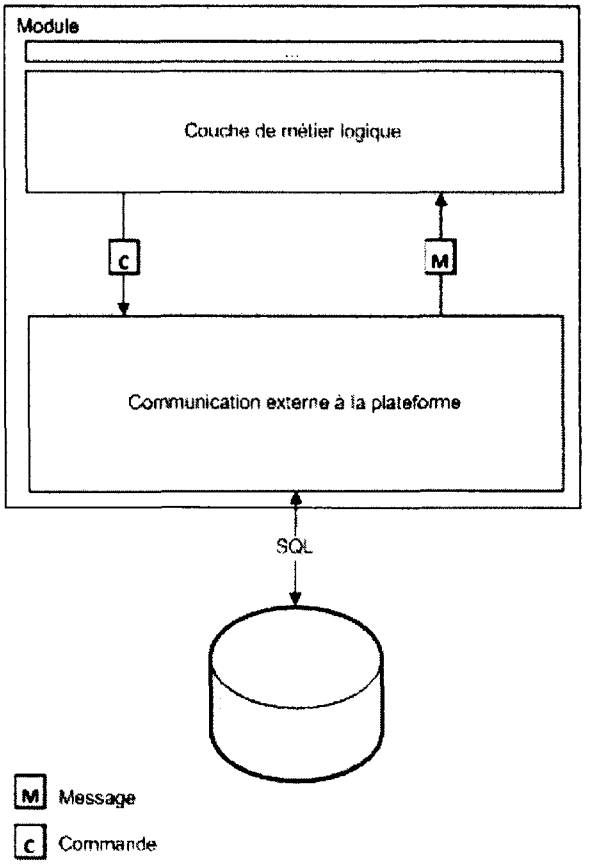

Figure 3.6 Exemple de communication externe au système

La trousse de développement n'offre aucun moyen de communication externe implémenté, car ces interactions sont propres à chaque module. Donc, lors de l'implémentation d'un module, si de tels mécanismes sont nécessaires, ils doivent être créés par le développeur. Par contre, la trousse offre deux mécanismes de communication implémentés et directement utilisables : JMS et les sockets en Java.

Les mécanismes implémentés

La trousse de développement implémente deux mécanismes de communication : le Java Messaging Service (JMS) [21] et les sockets [32]. Le tableau 3.2 présente une comparaison entre ces deux mécanismes de communication.

Tableau 3.2 Comparaison JMS et Socket

	JMS	Socket
Dépendances entre modules	Connaître le courtier	Modules doivent se connaître
Communication	Transige par le courtier	Entre les modules
Chiffrement	Si le courtier le supporte	Supporté

L'implémentation d'un mécanisme de communication utilisant l'API JMS permet de communiquer avec des courtiers de messages avec divers protocoles comme AMQP. En utilisant ce mode de communication, les dépendances entre les divers modules d'un système diminuent, puisque les modules n'ont pas à se connaître pour échanger des messages. Cependant, un désavantage de JMS est qu'il est dépendant des services supportés par le courtier. Par exemple, il existe des courtiers qui ne supportent pas le chiffrement des messages et ne garantissent pas la transmission de tous les messages. Lors du choix du courtier, ces caractéristiques devront être prises en compte.

Le deuxième mécanisme de communication implémenté dans la trousse de développement est les sockets sous le protocole TCP/IP. Les sockets sont un moyen efficace et simple de transférer de l'information entre divers modules distants. Cependant, l'utilisation des sockets apporte une plus grande dépendance entre les modules, car chacun des modules doit connaître le module avec lequel il veut communiquer. L'utilisation des sockets peut se faire sous deux modes, en mode non chiffré ou en mode chiffré sous SSL/TLS [31] avec authentification du serveur et ou client par certificat.

L'implémentation de ces deux mécanismes de communication amène de bonnes possibilités pour un module. L'utilisation de JMS avec un courtier de message comme point central diminue les dépendances entre chaque module et les sockets permettent une communication directe avec un autre module.

3.2.2 Couche de métier logique

La couche de métier logique est constituée d'analyseurs. Le rôle de ces analyseurs est de convertir les messages reçus de la couche d'accès à l'information en un type de données connu par le module et le convertir ensuite en un objet sérialisable.

Tel qu'expliqué précédemment, les analyseurs s'enregistrent à une communication afin de recevoir les nouveaux messages. Les gestionnaires de la couche supérieure (la couche de traitement) sont conçus sur le même principe, ils s'enregistrent auprès des analyseurs afin de recevoir les nouvelles données. En d'autres mots, l'analyseur observe les communications et est observé par le gestionnaire de la couche de traitement. Lors de la création d'un

analyseur, une communication lui est associée. Ceci lui permet d'envoyer des données vers l'extérieure du module. Un exemple de communication entre l'analyseur et les autres couches est présenté à la figure 3.7.

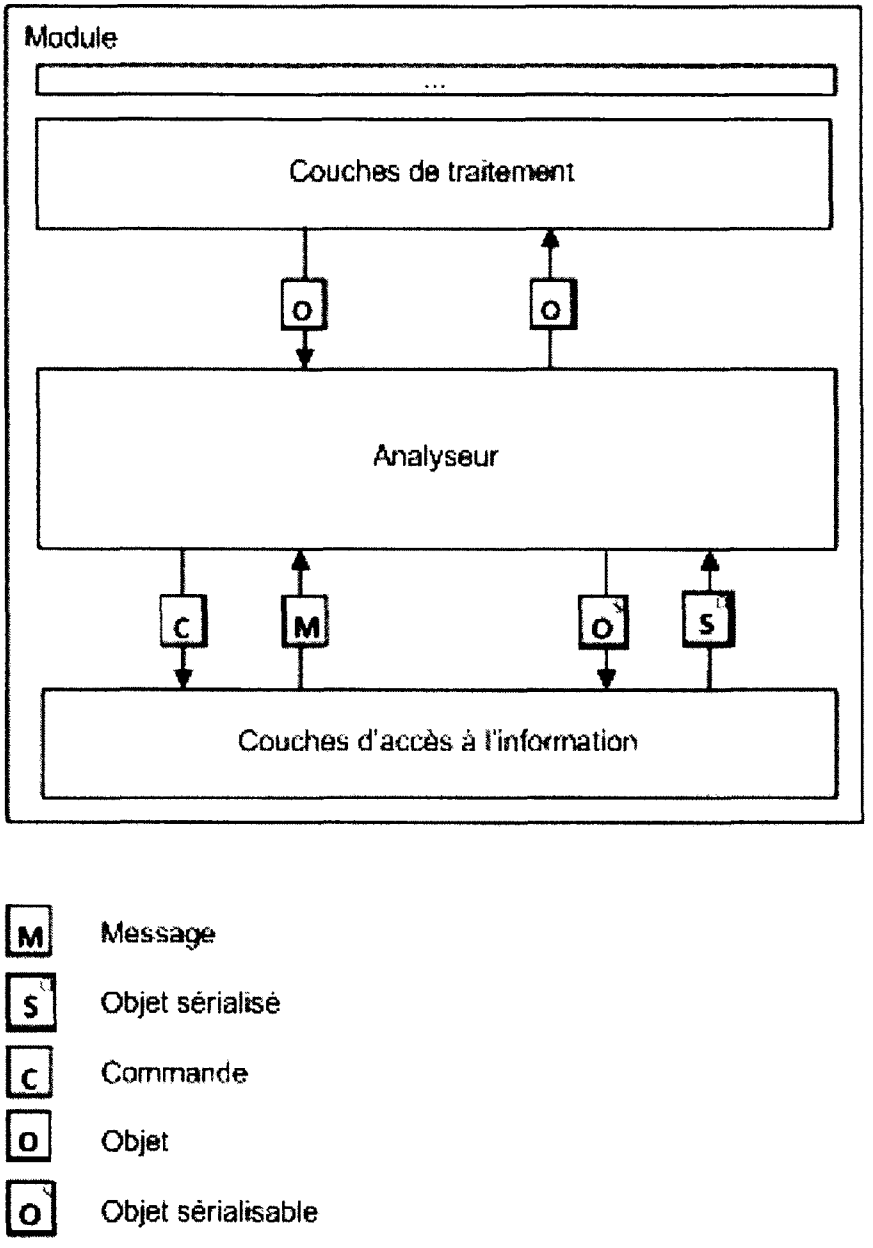

Figure 3.7 Exemple d'analyseur

La trousse de développement logiciel fournit deux analyseurs : un analyseur XML (Analyseur XML) et un analyseur à balancement de charges (Analyseur BC). D'autres analyseurs pourraient être nécessaires, principalement pour les mécanismes de communication externes qui sont uniques à chaque module.

Analyseur XML

L'un des analyseurs fournis dans la trousse de développement est un analyseur pour le langage de balisage extensible (XML). Afin de transmettre les données d'un module vers un autre, l'analyseur convertit les objets connus par le système en XML et lors de la réception de message XML, l'analyseur reconstruit ces messages sous forme d'objets connus par le module. De cette façon, tout module qui peut analyser du XML est en mesure de comprendre les messages qui sont émis par les modules utilisant ces analyseurs.

Afin d'accomplir cette tâche, l'analyseur XML utilise la bibliothèque XStream [48]. À l'aide de cette bibliothèque, il accomplit le transfert d'un objet vers XML et vice versa. Cette transformation est coûteuse lorsqu'elle utilise un convertisseur générique. Alors, afin d'augmenter la vitesse de conversion des objets, des convertisseurs spécifiques peuvent être développés.

Le sérialiseur XStream a été choisi parce qu'il est facile d'utilisation et est implémenté pour d'autres langages de développement. De cette façon, les dépendances avec Java sont diminuées. XML est une façon de représenter des données sous format texte donc, n'importe quel langage en mesure de traiter du texte sera en mesure de traiter du XML.

L'analyseur contient deux listes bloquantes : une première liste de traitement pour la conversion d'objet en XML et une deuxième pour accomplir l'inverse. Chacune de ces listes est accédée par un nombre de threads défini lors de la configuration du module. Ces listes sont de taille fixe, alors, lorsqu'elles sont pleines, les fonctions permettant d'ajouter dans celle-ci bloquent l'exécution jusqu'à ce qu'une place se libère. En limitant le nombre d'objets dans la liste à traiter, la couche assure de ne pas être dans un mode d'instabilité où le nombre d'objets à traiter augmente de façon plus rapide que la vitesse de traitement, ce qui risquerait de produire des erreurs de corruption de la mémoire.

Analyseur à balancement de charge

L'Analyseur BC est un analyseur qui permet d'utiliser «N» analyseurs pour convertir des objets en messages. Pour ce faire, lors de la création de cet analyseur, des analyseurs sont enregistrés auprès de celui-ci. Lorsque le gestionnaire envoie un message à cet analyseur, celui-ci distribue le message à l'un des analyseurs présents dans sa liste. Afin de choisir lequel des analyseurs recevra le message, le balanceur conserve le dernier analyseur à qui il a envoyé un message et prend le prochain dans sa liste.

Avec un tel analyseur, le système peut distribuer l'envoi de messages vers différents analyseurs donc diverses communications. Puisque chaque analyseur est lié à une communi-

cation, les données peuvent aisément être distribuées vers différents modules ou systèmes. Par exemple, si le balanceur comporte trois analyseurs XML qui sont chacun liés à leur socket, l'envoi de message pourra être envoyé en alternance vers trois modules différents.

Le rôle de cet analyseur est de permettre le contournement d'endroits où le traitement des données est plus laborieux en parallélisant le traitement à accomplir.

3.2.3 Couche du traitement

La couche de traitement est celle où les décisions sont prises dans un module. Cette couche est divisée en deux sous-couches : le gestionnaire et l'intelligence du module. De même que les couches précédentes, cette couche est construite selon un patron de conception observateur. Le gestionnaire observe l'analyseur de la couche de métier logique, la sous-couche d'intelligence observe le gestionnaire et cette dernière est observée par la couche de présentation si celle-ci est présente.

Afin de comprendre la relation entre ces deux sous-couches et le module, la figure 3.8 illustre les échanges de données se produisant à ce niveau. Le gestionnaire dirige les données provenant de la sous-couche d'intelligence vers les analyseurs enregistrés sur ce type de données. Il fait de même avec les données provenant des analyseurs, vers la sous-couche d'intelligence du module.

La sous-couche d'intelligence du module a pour rôle de traiter les données reçues et de prendre les décisions appropriées. C'est à cet endroit que se retrouvent les algorithmes d'un module.

Gestionnaire

Lors de l'implémentation de la trousse de développement logiciel, un gestionnaire a été implémenté. Il permet de rediriger les données basées sur leur type. Les analyseurs et la sous-couche d'intelligence s'enregistrent auprès du gestionnaire en lui spécifiant pour quel type de données ils désirent être informés et lorsque ce type de données est disponible, le gestionnaire les faits suivre à qui de droit. En d'autres termes, le gestionnaire joue le rôle de courtier de données local du module.

Puisque des requêtes proviennent de plusieurs endroits, un enjeu important est la gestion des threads. Afin de se protéger contre les problèmes futurs, chaque opération est faite de façon atomique. Chaque donnée est envoyée à son destinataire et ne peut être modifiée entretemps.

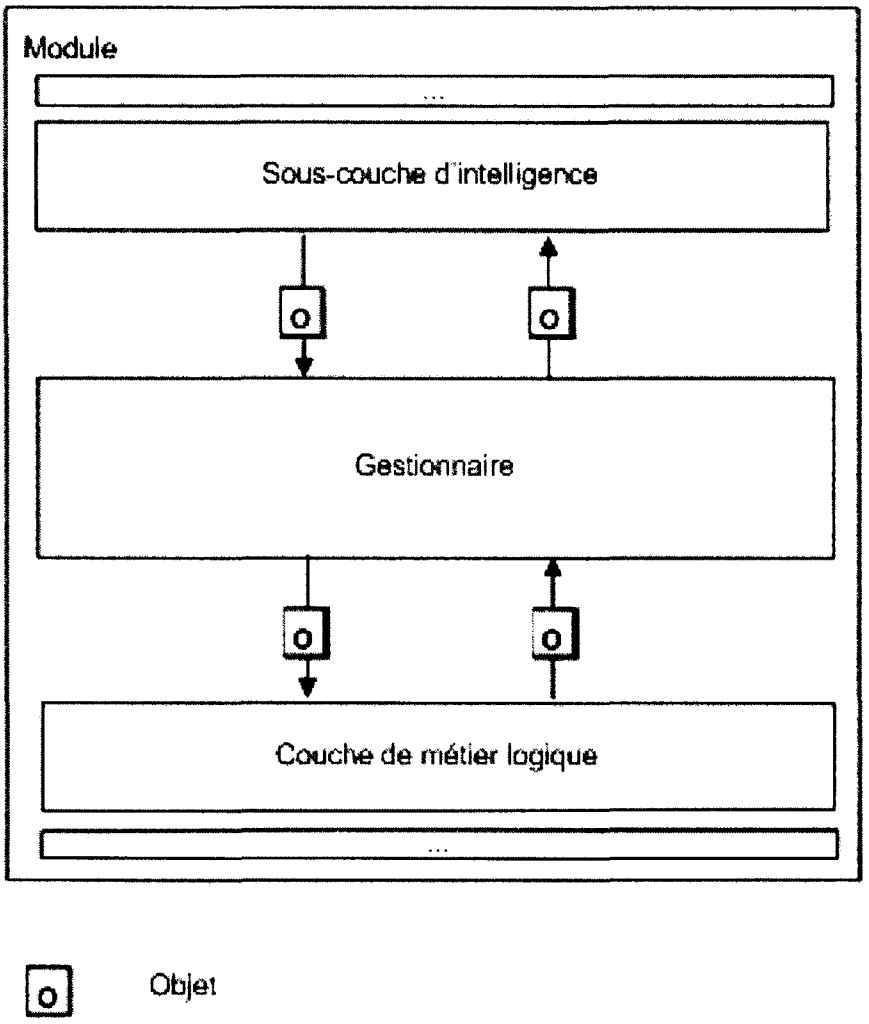

Figure 3.8 Exemple de gestionnaire

Lorsque des données entrent dans le gestionnaire, elles sont ajoutées à une liste bloquante. Par la suite, un fil d'exécution prend ces données et les distribue à qui est enregistré pour ce type de données. Donc, le gestionnaire comporte deux listes et deux threads, une section pour émettre des données des analyseurs vers la sous-couche d'intelligence et une section pour passer de la sous-couche d'intelligence vers les analyseurs. Il est important de noter que les listes utilisées sont de taille fixe, lorsque celles-ci sont pleines, les méthodes d'ajouts à ces listes bloquent jusqu'à ce qu'une place se libère. Ce mécanisme est mis en place afin de protéger le gestionnaire et empêcher la possibilité d'être dans un état instable où les données arrivent plus rapidement qu'elles peuvent être traitées.

Intelligence du module

Étant donné que le rôle de chaque module est différent, la sous-couche d'intelligence du module n'est pas implémentée. Cette section contient tous les algorithmes d'un module. Lors du développement d'un nouveau module, cette sous-couche devra être implémentée. Comme les autres couches, une abstraction de cette sous-couche est disponible afin de standardiser le développement.

3.2.4 Couche de présentation

Le rôle de la couche de présentation est d'afficher des informations à un utilisateur. Étant donné que la majorité des modules sont utilisés en mode service, cette couche n'est pas utile pour eux. Alors, l'implémentation de cette couche est optionnelle. Cette couche risque d'être utilisée majoritairement dans les modules de configuration de la plateforme.

Tout comme les autres couches, celle-ci est pensée pour fonctionner sous le patron de conception observateur. Celle-ci s'enregistre sur la sous-couche de l'intelligence du module.

3.2.5 Les mécanismes de configuration

Le but de ce projet est de concevoir une plateforme distribuée collaborative et l'un des objectifs est que cette plateforme soit configurable en temps réel. Afin d'atteindre ce requis, un système de configuration des modules a été mis en place.

Pour débuter, lors du démarrage d'un module, il doit être initialisé. Pour ce faire, un dossier de configuration est situé à la racine du module et contient les fichiers de configuration initiaux. Ceux-ci sont exécutés par ordre alphabétique par le module lors de son démarrage.

Par la suite, afin de configurer les modules lors de l'exécution de ceux-ci, un système basé sous le principe du patron de conception commande a été conçu. Une problématique est survenue, les différents modules ne connaissent pas les configurations possibles d'un module puisque chaque couche peut être implémentée différemment dans chaque module. Afin de résoudre cette problématique, lors de l'initialisation d'un module, il enregistre un exemple de configuration possible pour chacune de ces couches. Cette information est alors accessible si elle est demandée par un autre module.

Finalement, il est intéressant avant d'envoyer une nouvelle configuration à un module, de connaître la configuration actuelle de celui-ci. Lors de l'implémentation de chacune des couches, une méthode doit être implémentée qui retourne l'état présent de la configuration de celle-ci.

Lors d'envois de la commande pour retrouver les configurations possibles pour un module trois informations sont retournées : les fichiers de configuration initiale, les exemples de configuration possibles pour le module et l'état de la configuration présente de ce module.

Une fois la configuration choisie, le fichier de configuration est envoyé au module. Une réponse pour confirmer le statut de l'exécution de la commande est retournée à l'envoyeur de la commande.

3.3 Le courtier

Le choix d'utiliser un courtier de message comme point central pour la plateforme a pour but d'apporter une indépendance entre les divers modules. Chaque module doit se rapporter au courtier en s'y enregistrant et par la suite, il peut transférer des données par ce point. De cette façon, chaque module n'a pas à connaitre tous les modules du système pour communiquer. Ceux-ci s'enregistrent sur un type de message, lorsque le courtier reçoit des messages de ce type, il les redirige vers ces modules.

Pour le choix du bon courtier, quelques points sont à prendre en considération. Tout d'abord, la plateforme est développée en Java et Java intègre un service de messages (JMS), alors l'utilisation de ce service reste un atout. De plus, beaucoup de courtiers supportent plusieurs protocoles de communication, alors afin de diminuer les dépendances et de permettre à des modules écrits en d'autres langages d'interagir avec le système le nombre de protocoles supportés par le courtier peut influencer sur le choix de celui-ci.

Un facteur important pour un courtier de messages est sa performance, combien de messages peut-il traiter dans un certain laps de temps. Afin de valider cet aspect, des tests de charge doivent être faits.

De plus, afin de choisir un bon courtier, la validation des mécanismes de sécurité que celui-ci offre est importante. Certains protocoles tels que JMS n'offrent pas de mécanisme de sécurité intrinsèque. Alors, tout dépend des services que peut offrir le courtier. Les points à valider sont : la possibilité d'authentification des parties, modules et courtiers, la vérification de l'intégrité des messages envoyés et la possibilité de chiffrer les messages.

3.3.1 Les courtiers potentiels

Deux courtiers ont été testés, le premier est QPid d'Apache et le deuxième est ActiveMQ d'Apache. Puisqu'il est simple de changer de courtier, et cela de façon transparente pour le logiciel, les tests ont été effectués une fois que les modules de la plateforme ont été implémentés.

Les tests consistent à avoir un module qui publie des données en continu (le module de filtrage) sur le courtier et un module qui les lies en continu (le module de gestions des données). Un troisième module est inséré dans le test afin de fournir des données au module qui publie sur le courtier. Les trois modules utilisés sont le module sonde pour récupérer des données d'un fichier de journalisation, le fichier de filtrage qui transmet les données au courtier et le module de gestion des données qui lie ces données. Il est à noter que tous ces modules sont installés sur le même ordinateur que le courtier.

Les résultats des tests sont présentés dans le chapitre 5 Tests du système.

3.3.2 Le choix du courtier

Suite aux tests, le courtier choisi est ActiveMQ d'Apache. En plus de fournir la meilleure vitesse de traitement, il répond à d'autres critères importants comme la possibilité d'authentifier le client, la possibilité de chiffrer les données sous SSL, la possibilité d'utiliser un protocole de rattrapage des pannes (failover) et la possibilité d'être utilisé en mode distribué (cluster).

3.4 Les modules d'acquisition de données

Deux modules dans la plateforme ont accès à des données provenant de l'extérieur de ce système : le module sondes (qui accède à des fichiers de journalisation) et le module de gestion des données (qui accède à des entrepôts de données). Ces deux modules sont présentés dans les deux prochaines sections.

3.4.1 Module sonde

Le rôle du module sonde est de lire un fichier de journalisation issu des systèmes de surveillance de réseaux. Pour la validation du système présenté dans ce mémoire, des données issues du Service des Technologies de l'Information de l'Université de Sherbrooke (STI de l'UdeS) ont été utilisées. Lors de la lecture, il doit générer des objets compris par le système et les partager avec la plateforme.

Selon le nombre d'usagers des réseaux traités, la taille des fichiers de journalisation peut être importante. La performance de cet élément est donc de première importance. À titre d'exemple, les fichiers de journalisation pour une journée fournis par le STI de l'UdeS varient entre 15 et 20 Go. Un fichier de journalisation de 20 Go représente environ 110 millions de lignes. Alors, le module sonde doit être en mesure de convertir 110 millions de lignes en 110 millions d'objets et les envoyer, à l'intérieur d'une journée, aux autres modules. En effet, si l'ensemble des données n'était pas traité en 24 heures, le système prendrait du retard et éventuellement ne serait plus en mesure de traiter la totalité des données capturées.

Afin de transmettre les données vers le module suivant, le module de filtrage, la configuration utilisée pour ce module est l'utilisation de sockets. Ce module est exploré dans les prochaines sections.

3.4.2 Module de gestion des données

Le module de gestion des données a deux rôles, le premier est d'écrire les données journalières filtrées dans une base de données et le deuxième est de publier les réponses de requête de données qui lui sont demandées.

Afin de permettre une intégration facile avec divers entrepôts de données, la bibliothèque Hibernate [19] est utilisée. Cette bibliothèque permet de convertir les objets d'un système en données pour une base de données, ainsi que changer de fournisseurs de base de données de façon transparente pour l'application. Pour l'utilisation de modèles de données simples, Hibernate est performant. Cependant, lorsque des requêtes plus complexes sont requises, celles-ci peuvent être implémentées.

La base de données choisie est MySQL. Il est à noter que le serveur MySQL a été installé sur un ordinateur portatif pour simplifier la tâche (Ordinateur 1 qui est défini dans le chapitre de test), une machine virtuelle aurait aussi pu être utilisée pour ce test. Puisque le cadre du projet n'est pas de démontrer la possibilité de traitement d'une base de données,

cette architecture a été choisie. Il est certain que dans notre architecture la performance du serveur de base de données, ici MySQL, risque d'être un élément déterminent pour celle de l'ensemble du système. Dans un milieu de production, avec les ressources financières nécessaires, MySQL pourrait être distribué sur un groupe de serveur afin d'augmenter la performance de traitement.

Lors de l'initialisation de ce module, il s'enregistre auprès du courtier de messages afin de recevoir les données qu'il devra enregistrer dans la base de données. De plus, il s'enregistre pour écouter les requêtes des divers modules et donc pouvoir leur fournir les données qu'ils désirent.

3.5 Les modules de traitement

Lors de la conception de la plateforme, trois modules ont été pensés pour traiter les données. Ces trois modules sont : un module de filtrage, un module de collaboration et des modules d'analyses. Les deux premiers modules ont été implémentés dans le cadre de ce projet et le dernier sera implémenté dans les prochaines phases du projet.

3.5.1 Module de filtrage

Le but du module de filtrage est de réduire la quantité de données publiées sur le système. Pour ce faire, il filtre à l'aide d'une liste statique de caractéristique. Présentement, le filtre accepte de filtrer par adresse IP et port de communication.

Toutes les données dont les adresses IP et ports ne sont pas présents dans la liste du module de filtrage sont envoyées au courtier afin d'être distribuées aux autres modules.

Le module de filtrage s'approvisionne des données fournies par le module sonde. Alors, il doit être en mesure de traiter la même charge de données que le module sonde.

3.5.2 Module de collaboration

Le module de collaboration agit à l'inverse du module de filtrage. Il est possible de passer en mode filtre à collaboration lors de l'initialisation de la sous-couche d'intelligence du module. Alors, lorsque ce mode est activé au lieu de supprimer les données dont les adresses IP et ports figurent dans la liste de filtrage, il les transmet aux modules qui en font la demande. Le mode collaboration permet de répondre aux requêtes provenant d'autres systèmes.

Ce module peut être configuré afin d'écouter sur les divers liens de communications entre les modules et de transmettre cette information à qui les demande.

3.5.3 Modules d'analyses

Lors de ce projet, les modules d'analyses n'ont pas été implémentés. Ces modules seront conçus dans une phase ultérieure du projet. Cependant, pour ces modules, l'utilisation de système de calculs distribués fonctionnant en parallèle (Storm ou Hadoop avec l'algorithme MapReduce) peut être une solution performante pour accomplir cette tâche.

Le rôle de ces modules d'analyses est de premièrement détecter la présence d'une attaque logicielle (botnet) sur un réseau et par la suite, de tenter de retracer la provenance de cette attaque.

3.6 Les modules de configuration

3.6.1 Interface utilisateur

Le module de configuration développé est une interface utilisateur en mode ligne de commande. À l'aide des mécanismes de configuration fournis avec la trousse de développement logiciel, il est possible de retrouver l'information concernant les configurations de chaque module. Alors, même si le module de configuration ne connait pas les configurations spécifiques pour chaque module, il est en mesure de les retrouver en les interrogeant. Par la suite, celui-ci peut afficher les choix possibles de configuration de chacun des modules à l'utilisateur. Une fois les choix de configuration effectués, celui-ci les transmet aux modules concernés.

Les reconfigurations de chacun des modules peuvent se faire en temps réel. Les configurations pouvant être ajustées en temps réel peuvent être : activer un nouveau type de communication entre deux modules, activer du balancement de la charge, etc.

3.7 En résumé

Lors du développement de ce projet, trois choses ont été accomplies. Premièrement, la conception de la plateforme distribuée. Ceci a amené la nécessité de concevoir une trousse de développement logiciel pour faciliter le développement et l'uniformiser. Par la suite, l'implémentation de la plateforme distribuée. Pour ce faire cinq modules ont été implémentés en utilisant la trousse de développement logiciel : le module sonde, le module de

filtrage, le module de gestion des données, le module de collaboration et l'interface utilisateur. Il est à noter que peu de ligne de code Java est nécessaire pour implémenter chacun de ces modules, voir figure 3.3. Pour finir, suite à des tests de charges le courtier de messages a été choisi.

Tableau 3.3 Nombre de lignes de code Java par module

	Nombre de lignes de code Java
SDK	4339
Module sonde	2760
Module gestion de l'information	970
Module de filtrage	535
Module de collaboration	535
Interface utilisateur	1420
Total	10559

CHAPITRE 4

Généralisation de la trousse de développement logiciel (SDK)

Dans le chapitre précédent, la trousse de développement logiciel conçue à cet effet a été présentée. L'un des objectifs visés lors de la conception de cette trousse était de produire un outil générique qui puisse être réutilisé dans des projets très variés. Cette trousse de développement logiciel est un nouvel outil permettant d'aider le développement du système distribué collaboratif en proposant une architecture à respecter lors du développement de modules. Ce chapitre se penche sur des situations où cette trousse peut être utilisée et de quelle façon son utilisation serait bénéfique pour les concepteurs de ces systèmes. Pour ce faire, trois thèmes sont explorés soit : l'utilisation de la trousse en entreprise, l'utilisation dans des situations résidentielles et l'utilisation dans un contexte général.

4.1 En entreprise

L'utilisation de la trousse de développement logiciel est intéressante pour une entreprise. Lors de son utilisation, un patron d'architecture pour le développement d'un module est établi. Ce patron est basé sur des patrons de conceptions connus et testés par la communauté tels que : le patron de conceptions de l'observateur et le patron de conception de commande. Autres que le patron pour l'architecture, des outils sont présents afin de faciliter les communications inter-modules, pour collaborer en échangeant des données ou encore pour la configuration de ces modules en temps réel. L'un des outils est un routeur de méta-données, le gestionnaire de données. Ce type d'outils permet de diriger les divers types de données d'un système vers divers points de traitement. Avec ce système, l'entreprise aurait simplement à fournir les ressources nécessaires comme une base de données, des serveurs de stockage, etc. afin d'implémenter leur système distribué. Deux exemples de systèmes pouvant utiliser cette trousse de développement logiciel sont présentés ici : les systèmes de récupération de statistiques et des systèmes de calcul distribué.

4.1.1 Récupération de statistiques

Par exemple, le SDK pourrait être utilisé pour le développement d'un engin de collecte de statistiques. L'architecture déjà construite sauvera du temps précieux lors du développement de tels systèmes, de plus il fournira des mécanismes essentiels dans ce type de logiciel comme la distribution de données, le balancement de la charge, la gestion décentralisée, la configuration dynamique, etc. La quantité de données à traiter pour ces applications peut être très grande. Alors, un système distribué collaboratif modulaire peut apporter des gains en performance et en maintenance à une entreprise.

Puisque la trousse de développement logiciel présentée est composée d'un gestionnaire de données qui agit comme un routeur de metadata, il est simple de distribuer les données recueillies vers différents points de traitement et ainsi d'apporter des gains en performance au système développé. De plus, telle que mentionné précédemment, l'architecture proposée offre la possibilité d'implémenter un système sous une architecture hiérarchique, ce qui apporte un facteur d'extensibilité important au système développé.

De plus, l'architecture facilite la modularité d'une application ce qui permet l'extensibilité de ce type de système. En mettant en oeuvre chaque tâche dans un module, il est simple de concevoir de nouveaux modules venant s'attacher au système pour effectuer différentes analyses sur les données. Donc, lorsque de nouvelles idées sont présentées, l'ajout d'un module au système pour répondre à ces fonctionnalités peut être implémenté rapidement sans affecter le système.

Avec les capacités de traitement et d'extensibilité présentées dans le prochain chapitre (Test du système), le SDK peut s'adapter pour traiter de très grandes charges de données, ce qui en fait une trousse de développement logicielle parfaite pour ce type d'application.

4.1.2 Calcul distribué

En entreprise, divers calculs peuvent être nécessaires et certains d'entre eux demandent de grandes quantités de ressources afin de les effectuer. La trousse de développement logiciel peut apporter une solution à cette problématique. Lors du développement d'une application, le temps consacré devrait être investi sur la conception des algorithmes et non pas sur la distribution du calcul. Afin de concevoir un système de calcul distribué, deux modules exploitant la trousse de développement logiciel sont nécessaires. Le premier est un gestionnaire qui détermine quelle section du calcul est à compléter et un autre qui effectue ce calcul. Une fois ces deux modules implémentés, le premier peut être exécuté avec balancement de la charge et démarrer le deuxième « N » fois sur divers ordinateurs de calculs.

De cette façon, la commande de calcul émise par le gestionnaire sera automatiquement distribuée parmi les différents modules enregistrés auprès du premier module.

Donc, pour les développeurs, seulement un algorithme de division du calcul et un algorithme de traitement des calculs doivent être implémentés, puisque les mécanismes de communication, les mécanismes de balancement de la charge et le patron de l'architecture sont déjà présents dans la trousse de développement logiciel.

De plus, une autre méthode d'utilisation de ce SDK pour accomplir des calculs distribués serait de l'utiliser comme gestionnaire de calcul intégré à un système Hadoop et MapReduce. Pour ce faire, les modules développés pourraient avoir pour rôle de recueillir des données de divers endroits et de les acheminer à un contrôleur. Celui-ci pourrait par la suite, à l'aide de l'algorithme MapReduce sur un système distribué Hadoop, accomplir les calculs en un temps raisonnable. Enfin, le module du système pourrait acheminer la réponse du système de calcul parallélisé et la redistribuer aux modules du système les désirant. Donc, en couplant le système développé, la trousse de développement logiciel et les principes de parallélisation de MapReduce, le système deviendrait très performant dans la réalisation de calculs.

4.2 Résidentiel

Lors du développement d'applications pour le domaine résidentiel, un aspect important à prendre en compte est la simplicité. Puisque les modules peuvent se reconfigurer entre eux sur demande suite aux facilités offertes par la trousse de développement logiciel, il est possible de distribuer un logiciel dans une résidence en demandant un minium d'effort au client. Une des applications envisageables pour cette trousse dans le milieu résidentiel est la domotique.

4.2.1 Domotique

La domotique est de plus en plus présente dans les milieux résidentiels. Un aspect intéressant dans ces systèmes est de permettre aux clients de configurer leurs systèmes tels qu'ils le désirent et de leur permettre d'ajouter ou de supprimer des fonctionnalités sur demande. La trousse de développement logiciel proposé répond à cette problématique.

Afin d'arriver à cet objectif, le SDK offre des mécanismes pour configurer des modules avec des paramètres de configuration inconnus. Comme expliqué précédemment, à l'aide de requêtes, il est possible de retrouver les paramètres de configuration d'un module. Alors,

avec un utilitaire, il est simple d'afficher les différentes configurations à l'utilisateur avec les valeurs par défaut de celles-ci.

En conséquence, un système développé à l'aide de cette trousse offrirait la possibilité pour le client de se procurer des modules de domotiques de diverses entreprises qui pourraient être agencés les uns avec les autres.

4.3 Dans un contexte général

Dans un contexte général, l'un des facteurs clés à considérer lors du choix d'une trousse de développement logiciel est sa simplicité d'utilisation. Le chapitre précédent a démontré cette caractéristique en plus de démontrer les avantages que cette trousse offre tels que : un gestionnaire de données basé sur le type, des possibilités de balancement de la charge, l'extensibilité d'un système utilisant cette trousse et la facilité d'un système à être configuré en temps réel.

Ces caractéristiques dans un contexte général sont utiles à plusieurs types de logiciel par exemple : des systèmes de sécurités informatiques, des systèmes d'analyses financières, etc.

4.3.1 En sécurité informatique

Comme démontré dans le chapitre précédent, la trousse de développement logiciel a facilité le développement des divers modules. Cette trousse définit une architecture à respecter pour chaque module, en fournissant des mécanismes de configuration des modules et en facilitant les échanges entre les modules.

Dans le domaine de la sécurité informatique, un aspect à considérer est la tâche colossale de l'analyse due à la quantité de données. Comme démontré dans le chapitre suivant, ce SDK a su répondre à cette tâche en distribuant les données aux divers modules dans le système de façon efficace et rapide.

Puisque le SDK facilite la modularité d'un système, de nouvelles fonctionnalités pourront être ajoutées dans le système, lorsque nécessaires. Cette caractéristique est importante pour être à jour, vu que le domaine de la sécurité informatique évolue très rapidement.

4.3.2 Système financier

Une autre application possible pour le SDK est dans le domaine des systèmes financiers. Beaucoup de logiciels de traitement des données financières sont développés afin de déterminer les meilleures possibilités d'investissement.

Afin d'utiliser un maximum de données pertinentes, un nombre important de traitements doivent être accomplis. Alors, l'utilisation de ce SDK pourrait aider à la réalisation de tels systèmes. Les outils présents dans le SDK tels que : le gestionnaire de données et les outils de balancement de la charge s'avèreraient utiles pour traiter cette importante quantité de données.

De plus, l'architecture proposée par le SDK offre la possibilité de diviser un système en module, ce qui permet son extensibilité. Alors, pour de tels systèmes, il serait avantageux d'utiliser cette trousse puisque plusieurs modules pourraient se rattacher au système tels que : module de capture de l'information, module d'analyse et module de décision.

4.4 En résumé

La trousse développée lors de ces travaux avait pour but de faciliter le développement de la plateforme en fournissant un modèle d'architecture pour les modules de plus que fournir des outils générique pour ceux-ci. Cependant, puisque le SDK a été développé de façon générique et réutilisable, de nombreux projets et types de projets pourraient bénéficier de son utilisation pour produire des applications distribuées rapides, sécuritaires et faciles à analyser, développer et modifier.

CHAPITRE 5

Tests du système

Ce chapitre présente les résultats des tests obtenus avec la plateforme distribuée développée. Quatre sections sont présentées : l'architecture de test, les tests effectués, l'analyse des tests et la flexibilité du système.

5.1 Architecture de tests

Afin d'accomplir les tests, deux ordinateurs ont été nécessaires ainsi qu'un disque dur externe. L'architecture est présentée à la figure 5.1.

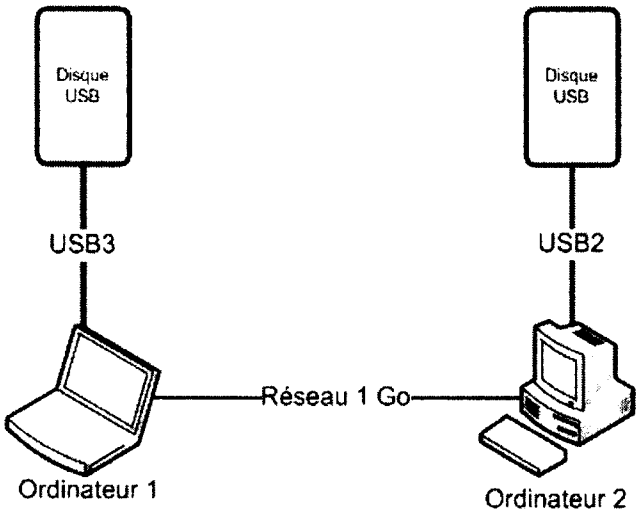

Figure 5.1 Architecture de tests utilisés

Dans ce modèle, trois composants sont utilisés : l'ordinateur 1, l'ordinateur 2 et un disque dur USB, le même disque dur est utilisé pour l'ordinateur 1 et l'ordinateur 2. Le tableau 5.1 présente les caractéristiques des deux ordinateurs et le tableau 5.2 présente celles du disque dur USB.

5.2 Présentation des tests

Cette section présente sept tests effectués sur la plateforme pour démontrer les performances offertes par celle-ci et identifier les éléments qui limitent la capacité de traitement du système.

Tableau 5.1 Caractéristiques des ordinateurs pour les tests

	Ordinateur 1	Ordinateur 2
Processeur	Intel I7 Q740 1.73 GHz (Quatre coeurs Hyperthreader)	Intel Pentium D 2.80 GHz (Deux coeurs)
Mémoire	8 Go	4 Go
Disque Interne	500 Go SATA2 - 7200RPM	160 Go SATA - 7200RPM
Latence moyenne disque	4.17 ms [37]	-
Vitesse carte réseau	Go	Go
USB	3.0	2.0
Système d'exploitation	Windows 7	Windows 8
Java	1.7	1.7
Base de données	MySQL 5.8	-
Courtier	ActiveMQ 5.8	-

Tableau 5.2 Caractéristiques du disque USB

	Caractéristiques
USB	2.0 et 3.0
Connexion interne	SATA3
Disque interne	OCZ Vertex 4
Type	SSD
Capacité	128 Go
Lecture séquentielle	560 Mo/sec [30]
Latence	0,2 ms [30]

Afin de tester le système, les données utilisées proviennent du service des technologies de l'information de l'Université de Sherbrooke. Le STI de l'UdeS a fourni divers fichiers de journalisation, ceux-ci peuvent atteindre des tailles de 20 Go de données par jour ce qui représente environ 110 millions de lignes. L'information disponible dans ces fichiers est : l'adresse IP source et destination, le port source et destination, l'heure du début de la connexion, la durée de la connexion, la taille des données transférées ainsi que la raison de fermeture de la connexion.

5.2.1 Performance de lecture d'un fichier de journalisation

Pour valider la performance en lecture d'un fichier de journalisation, deux tests ont été effectués sur l'ordinateur 1. Le premier consistait à lire le fichier de journalisation du disque dur local de l'ordinateur, le deuxième consistait à le lire du disque dur USB voir figure 5.2. Les résultats obtenus sont présentés au tableau 5.3. Ce test démontre que l'utilisation du disque dur USB offre une rapidité en lecture supérieure de 7,56% par rapport au disque local de l'ordinateur 1. Avec les ressources disponibles pour ces tests, ce nombre n'est pas

significatif puisque la lecture du fichier n'est pas le goulot du système. Cependant, dans un système où plus de ressources seraient disponibles pour effectuer le traitement, cette valeur pourrait être très importante, puisque la lecture du fichier pourrait devenir le goulot du système.

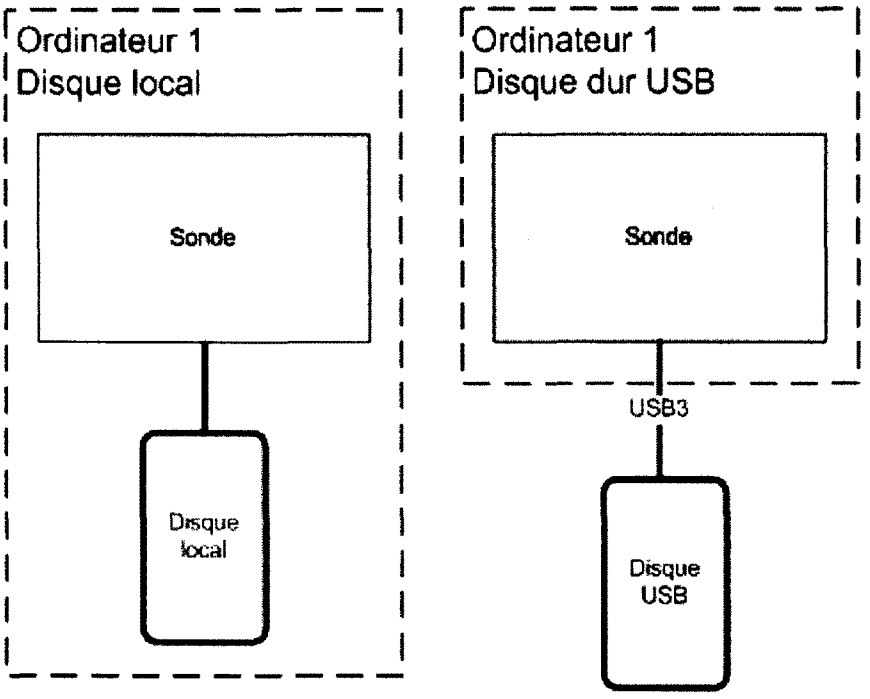

Figure 5.2 Architecture du test de la lecture de fichiers de journalisation

Tableau 5.3 Lecture de fichiers de journalisation

	Traitement (lignes / seconde)
Disque local	56 130
Disque dur USB	60 722

5.2.2 Performance de communication entre le module sonde et le module filtre

Afin de valider les performances de communication entre le module sonde et le module de filtrage, un test de transfert de message a été effectué. Pour accomplir ce test, le module sonde et le module de filtrage ont été installés sur l'ordinateur 1, les données étaient lues par le module sonde et elles provenaient du disque dur USB et envoyées par socket au module de filtrage voir figure 5.3. Les résultats sont présentés au tableau 5.4. Ce test a eu lieu pour montrer les effets du filtre sur la rapidité du système. L'analyse des résultats sera présentée à la section 5.3.

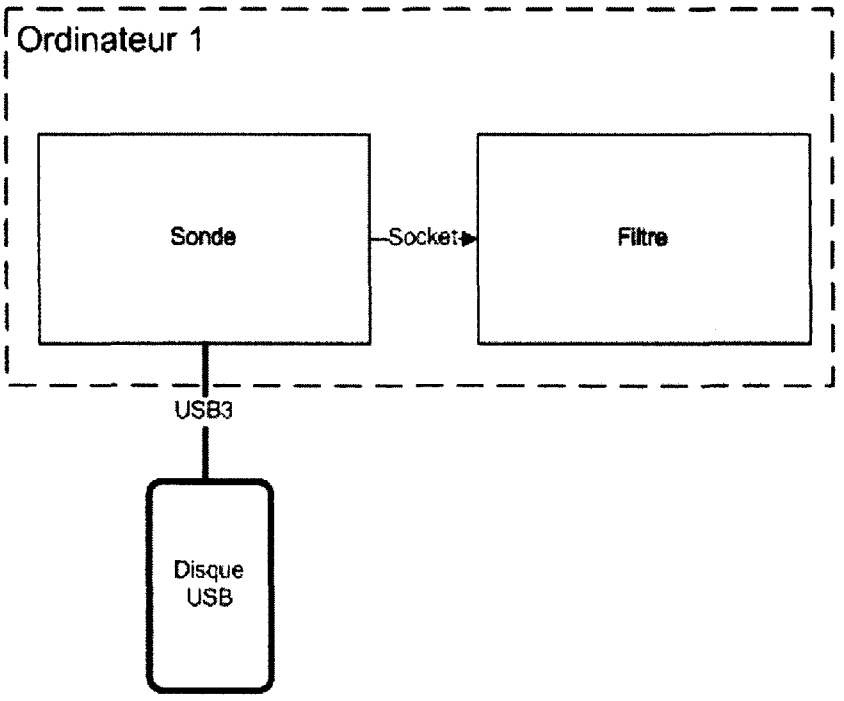

Figure 5.3 Architecture du test de la communication entre le module sonde et le module filtre

Tableau 5.4 Communication entre le module sonde et le module filtre

	Traitement (messages / seconde)
Test de communication	23 223

5.2.3 Performance de communication du courtier de messages

Deux courtiers ont été identifiés auparavant, QPid d'Apache et ActiveMQ d'Apache. Ce test a pour but d'identifier le plus performant des deux. Le test a été réalisé sur l'ordinateur 1 et consistait à lire les données à l'aide du module sonde et les transférer au module de filtrage. Ce dernier les envoie au courtier qui les redirige vers le module de gestion des données voir figure 5.4.

Ces tests ont été effectués avec des variantes. Le premier test a été accompli avec une connexion vers le courtier et le deuxième test avec trois connexions distinctes balancées vers le courtier. Les résultats sont présentés au tableau 5.5. Ces tests démontrent que le courtier ActiveMQ est plus rapide que QPid.

De plus, les tests utilisant trois connexions balancées vers le même courtier sont beaucoup plus rapides que les tests utilisant une simple connexion. Les performances des deux courtiers avec de multiples connexions sont très similaires. Lorsqu'il y a plusieurs connexions, le goulot n'est donc pas le traitement fait par la bibliothèque localement, mais le traitement

accompli sur le serveur, ce qui explique des performances similaires, lors de l'utilisation de plusieurs connexions.

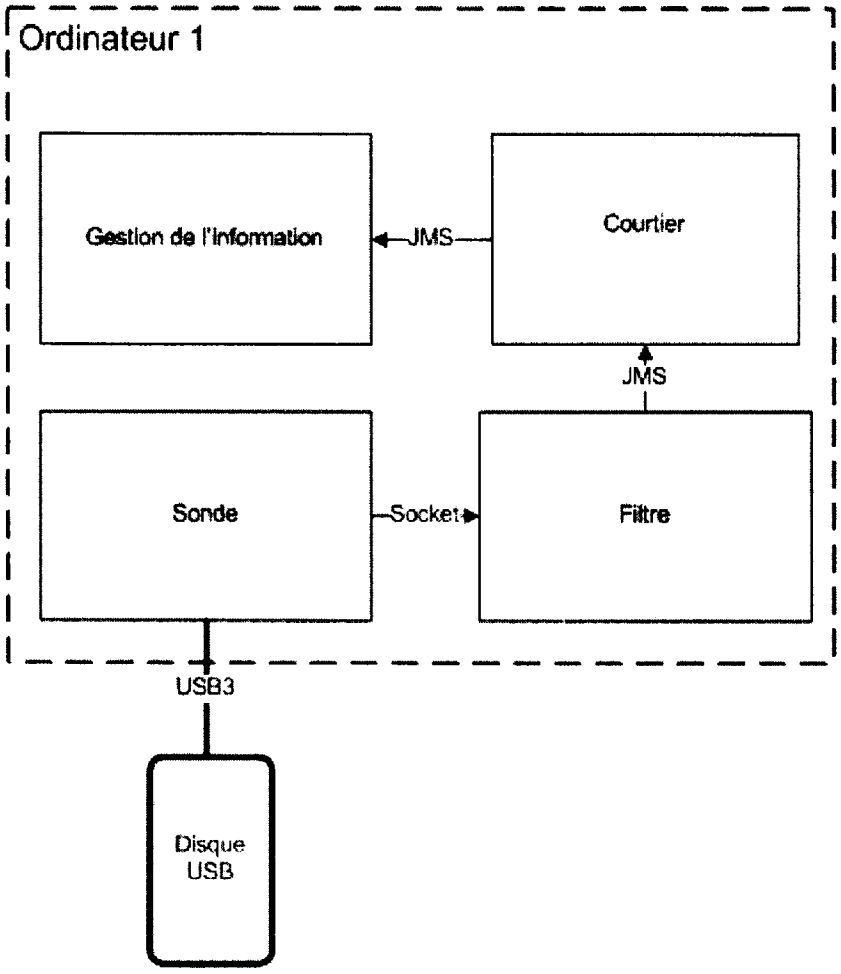

Figure 5.4 Architecture du test de la performance de communication du courtier de messages

Tableau 5.5 Performance de communication du courtier de messages

	Non balancé (messages / seconde)	Balancé (messages / seconde)
ActiveMQ	8 095	11 283
QPid	6 864	11 111

5.2.4 Performance d'écriture dans la base de données

Ce test avait pour objectif de valider la performance du système lors de l'écriture dans la base de données. Pour ce faire, tous les modules sont installés sur l'ordinateur 1 : le module sonde, le module de filtrage et le module de gestion de l'information. De plus, le courtier ActiveMQ et le serveur de base de données MySQL le sont aussi. Pour ce qui est des données, elles proviennent du disque dur USB.

Le test consiste à lire les données avec le module sonde, les envoyer au module de filtrage, les transmettre au courtier qui les redirige vers le module de gestion de l'information et ce dernier les enregistre dans la base de données voir la figure 5.5.

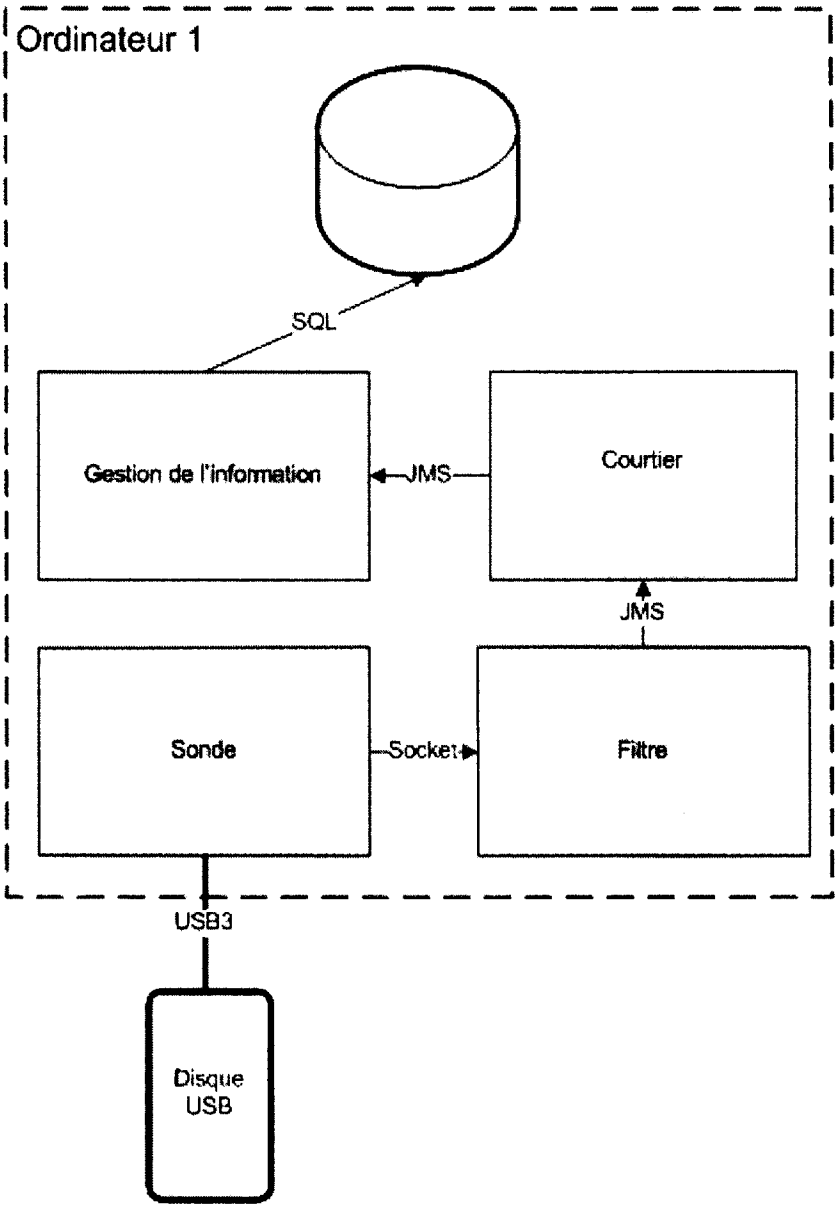

Figure 5.5 Architecture du test de l'enregistrement de données dans la base de données

Cinq essais ont été effectués pour déterminer le meilleur moyen d'enregistrer dans la base de données.

1. Conserver les configurations par défaut de Hibernate en envoyant les requêtes à la base de données pour chaque message.

2. Utiliser les configurations par défaut de Hibernate, cependant il enregistre dans la base de données par lot de 1000 messages. Donc, chaque enregistrement comporte 1000 messages.

3. Utiliser une connexion sans état et comme dans le deuxième essai enregistrer dans la base de données par lot de 1000 messages.

4. Utiliser les mêmes configurations que l'essai trois avec une variante : l'engin de la base de données n'est plus celui par défaut, mais plutôt MyISAM [28].

5. Utiliser le JDBC sans passer par Hibernate, en utilisant des requêtes préparées (prepared statement) sous l'engin MyISAM.

Les résultats sont présentés au tableau 5.6.

Tableau 5.6 Enregistrement de données dans la base de données

	Traitement (messages / seconde)
Essai 1	11
Essai 2	21
Essai 3	827
Essai 4	1859
Essai 5	1902

5.2.5 Performance de communication du courtier de messages de façon distribuée

Afin d'alléger l'ordinateur 1 de toutes les tâches, un deuxième ordinateur a été introduit au système, l'ordinateur 2. Ce test évalue les performances de communication entre l'ordinateur 1 et l'ordinateur 2. Pour ce faire, ActiveMQ et le module de gestion de l'information sont installés sur l'ordinateur 1. Le module sonde et le module de filtrage sont installés sur l'ordinateur 2. Trois tests sont effectués afin de déterminer le type de communication optimale : une connexion vers le courtier, trois connexions balancées vers le courtier et six connexions balancées vers le courtier voir figure 5.6. Les résultats sont présentés au tableau 5.7. En référence à ces résultats, les trois connexions balancées s'avèrent plus efficaces.

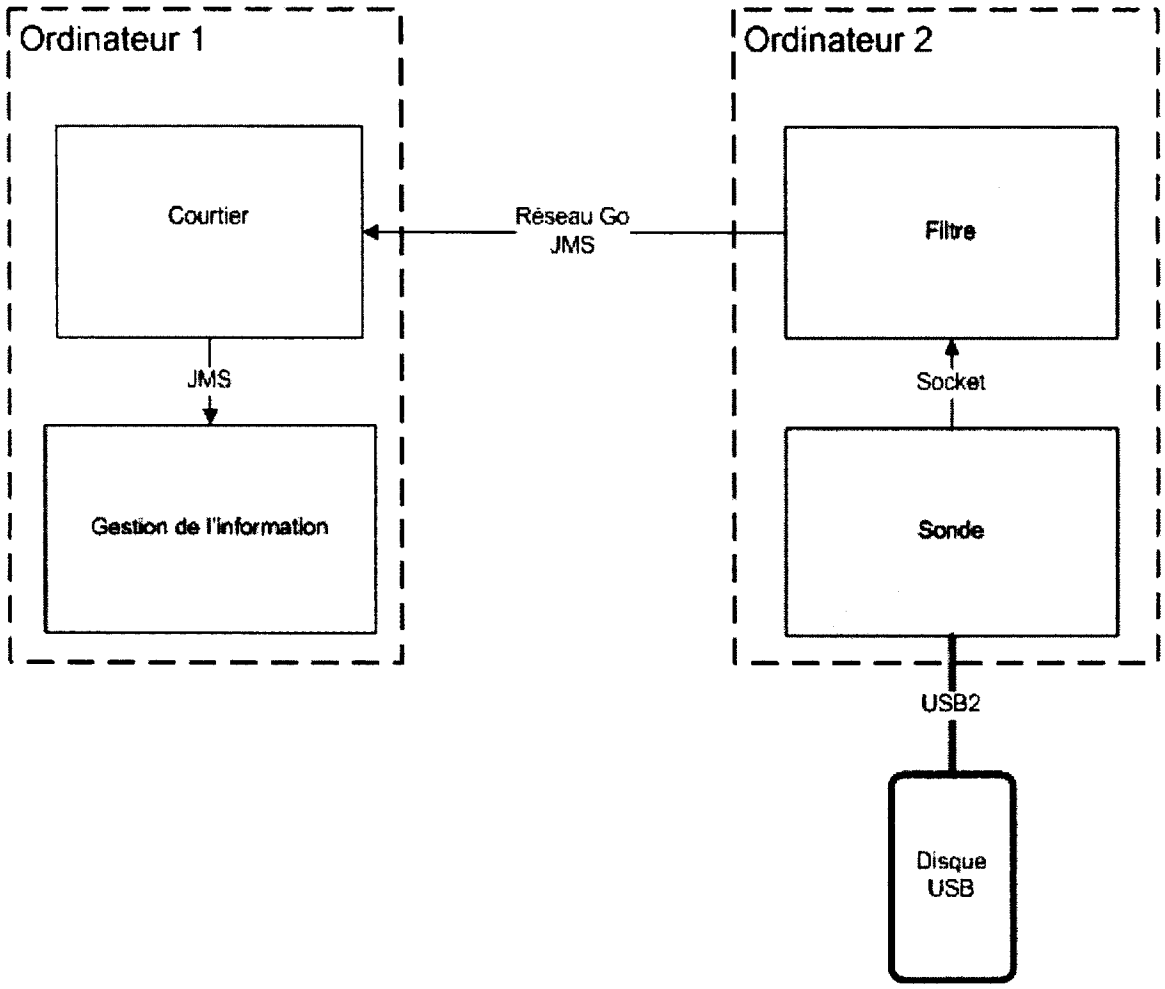

Figure 5.6 Architecture du test de la performance de communication du courtier de messages de façon distribuée

Tableau 5.7 Performance de communication du courtier de messages de façon distribuée

	Traitement (messages / seconde)
Une connexion	1 125
Trois connexions	2 239
Six connexions	2 214

5.2.6 Performance d'écriture dans la base de données de façon distribuée

Le test suivant a été effectué pour valider qu'il est possible d'améliorer l'accès à la base de données en diminuant la charge de travail de l'ordinateur 1 afin qu'il puisse accorder plus de ressources au serveur de base de données. Comme le test précédent, le module sonde et le module de filtrage sont installés sur l'ordinateur 2. Le module de filtrage communique

avec le courtier sur l'ordinateur 1 à l'aide de trois connexions balancées, le courtier redirige les données vers le module de gestion de l'information qui est sur le même ordinateur et pour finir ce dernier enregistre les données reçues dans la base de données voir la figure 5.7. Le premier test est effectué avec les configurations par défaut de Hibernate. Le deuxième essai sauvegarde les données tous les 1000 messages avec l'engin de la base de données qui est MyISAM. Les résultats de ces tests sont présentés au tableau 5.8.

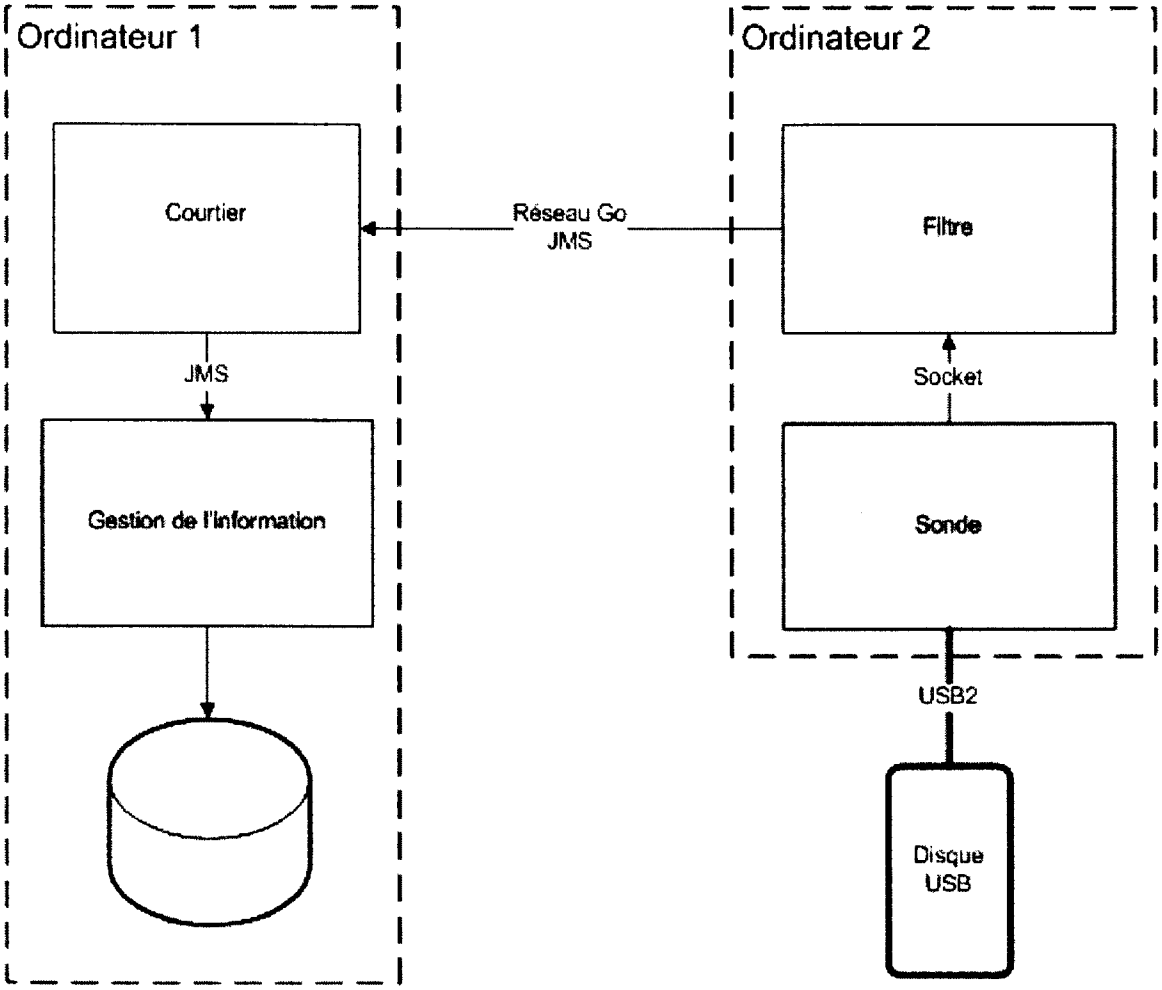

Figure 5.7 Architecture du test de la performance d'écriture dans la base de données de façon distribuée

Tableau 5.8 Performance d'écriture dans la base de données de façon distribuée

	Traitement (messages / seconde)
Essai 1	21
Essai 2	2307

5.2.7 Traitement complet d'un fichier

Afin de valider le système sur une longue période de temps et de démontrer sa stabilité lorsqu'il gère de grandes charges de donnée, le test final consiste à analyser les fichiers en entier fournis par le STI de l'UdeS. Pour accomplir ce test, l'architecture choisie est la même qu'au troisième test voir figure 5.4, tous les modules sont installés sur l'ordinateur 1 avec l'utilisation du disque dur USB pour la lecture des données. Les résultats sont présentés au tableau 5.9.

Tableau 5.9 Traitement complet d'un fichier

	Nombre de lignes	Temps d'exécution	Taille du fichier	Traitement
1	86 000 000	118 minutes	14,8 Go	12093 messages / seconde
2	112 000 000	130 minutes	19,3 Go	14317 messages / seconde

5.3 Analyse des résultats

Les tests ont permis de choisir la configuration optimale avec les ressources disponibles. Le premier test démontre que lire les données du disque dur USB est plus efficace que les lire du disque local de l'ordinateur 1 de 7.6%. Par la suite, le troisième test permet de choisir le courtier ainsi que son mode de connexion. ActiveMQ est choisi avec trois connexions balancées pour s'y connecter. Ensuite, le troisième test permet de trouver la meilleure configuration pour l'enregistrement dans la base de données. La configuration choisie est Hibernate avec une publication des résultats tous les 1000 messages avec une base de données sous un schéma MyISAM. Finalement, le sixième test permet de voir que le système peut être optimisé en distribuant la charge, la vitesse de traitement a passé de 1859 messages par seconde à 2214, 16% plus rapide.

Le système a démontré ses performances avec la possibilité de distribuer 11283 messages par seconde aux différents modules enregistrés auprès du système. Afin de traiter le fichier de taille maximale fournie par le STI de l'UdeS, 20 Go de données comportant 110 millions de lignes, le système doit traiter un minimum de 1273 lignes par seconde. Les tests réalisés ont démontré que ce niveau de performance est atteint. De plus, le cinquième test démontre que la distribution des communications sur plusieurs ordinateurs est efficace en validant la capacité de traiter 2239 messages par seconde. Afin d'offrir plus de performance, le serveur de courtier (ActiveMQ) et le serveur de base de données (MySQL) devraient être installés sur une infrastructure physique puissante.

Les tests ont démontré que chaque module ajouté au traitement apporte une diminution de la performance, ce qui est montré au tableau 5.10.

Tableau 5.10 Résumé des résultats

	Traitement (messages / seconde)	Perte (%)
Lecture fichier de journalisation	60 722	-
Module de filtrage	23 223	62
Module de gestion	11 283	51
Enregistrement de données	1859	83

Pour compléter, le septième test démontre le fonctionnement du système sur une longue période de temps. En traitant 112 millions de lignes d'un fichier de journalisation et en distribuant ces messages aux divers modules du système.

5.4 Flexibilité du système

La flexibilité du système a été démontrée lors des tests présentés. En modifiant quelques lignes dans les fichiers de configuration, le système passe d'un système à connexion unique à un système balançant la charge de travail vers plusieurs points. De plus, toujours en modifiant ces fichiers de configuration, il est possible de changer les destinations des connexions pour permettre aux modules d'être distribués sur plusieurs ordinateurs. Enfin, pour réaliser les tests, des connexions entre les différents modules ont été ajoutés et enlevés, ainsi que l'activation et la désactivation de l'enregistrement dans la base de données. Toutes ces configurations sont faites en modifiant quelques lignes dans des fichiers XML.

De plus, comme démontré dans le chapitre précédent, le rôle du module de configuration est l'ajustement en temps réel des paramètres système. Alors, une alternative à modifier manuellement les fichiers de configurations pour tester chaque section de la plateforme aurait été de le faire à l'aide de ce module.

Pour utiliser l'avantage de la flexibilité dans le système développé, les modules d'analyses qui seront développées ultérieurement pourront utiliser les mêmes mécanismes que le module de configuration afin de rediriger les liens entre les modules vers des endroits stratégiques ou encore changer les options sur le type de données à distribuer dans le système.

5.5 En résumé

En résumé, les tests démontrent que le système est en mesure de traiter les données et de les distribuer à différent module de traitement dans un temps raisonnable. De plus, lors de ces tests, les modules ont été reconfigurés de diverse façon afin de permettre de tester chacune des sections du système, tout cela accompli en modifiant quelques lignes dans des fichiers ce qui démontre la flexibilité et l'adaptabilité du système.

CHAPITRE 6

Conclusion

Trois axes seront développés : un sommaire du projet, les contributions apportées à la construction d'un outil permettant l'analyse d'attaques logicielles ainsi que les travaux futurs potentiels.

6.1 Sommaire

Lors de ce projet, la première réalisation a été de présenter une architecture distribuée collaborative. Celle-ci est divisée en cinq modules : le module sonde, le module de filtrage, le module de gestion de l'information, le module de configuration et le module de coordination. Chacun de ces modules a une fonctionnalité bien établie, de façon à ce que la plateforme soit extensible, flexible et facile à maintenir. Lors de la conception de la plateforme, deux éléments ont été sélectionnés parmi des services déjà existants : le point central du système, le courtier de message ActiveMQ d'Apache et le serveur de base de donnée MySQL.

Lors de l'implémentation de cette plateforme, une trousse de développement logiciel générique a été développée. Celle-ci a permis de faciliter le développement des modules en fournissant un patron d'architecture à respecter lors de son développement. De plus, le SDK est composé d'outils génériques qui ont facilité le développement des principales composantes du système comme : mécanismes de communication (Socket, JMS), analyseurs (XML, balancement de la charge), mécanismes de configuration ainsi qu'un gestionnaire basé sur le type de données.

Cette trousse de développement logiciel a eu pour rôle de faciliter le développement de ce système. Lors de son implémentation, des efforts ont été mis afin qu'elle soit réutilisable dans d'autres projets qui nécessiteraient une plateforme distribuée.

Pour finir, les tests ont démontré les performances du système. Celui-ci est en mesure de distribuer et traiter 11 283 messages par seconde. Pour ce faire, il les envoie au module de filtrage qui les transmet au courtier de messages. Le goulot est présentement l'écriture dans la base de données qui réduit les performances du système à 1859 messages par seconde,

ce qui est supérieur au nombre minimal de messages à traiter pour un traitement en temps réel pour un réseau de l'envergure de l'UdeS.

6.2 Les contributions apportées

Le but global du projet est d'implémenter une plateforme distribuée collaborative permettant de retracer la provenance d'une attaque de botnet. Plus précisément, ce projet porte sur le développement de la plateforme distribuée collaborative qui permettrait de collecter un maximum de données pertinentes pour permettre de retracer la provenance d'une attaque logicielle. La contribution principale de ce travail est : une plateforme distribuée collaborative flexible qui est en mesure de traiter une grande quantité de données et qui peut être reconfigurée en temps réel afin de faciliter la détection d'un botnet ainsi que la source d'une attaque distribuée logicielle.

Présentement, la plateforme est en mesure de collecter une grande quantité de données, de les prétraiter avec un filtre sommaire filtrant sur les adresses et ports de communications, de les distribuer aux modules les désirant et de les enregistrer dans une base de données. L'infrastructure est prête à interagir avec des modules d'analyses qui permettront de détecter un botnet et par la suite de retracer la provenance d'attaques logicielles. De plus, des mécanismes de collaboration ont été mis en place comme rediriger les liens de communication vers différents points dans le système, reconfigurer le filtre pour être plus ou moins permissifs, envoyer des requêtes aux modules de gestion pour retrouver des données conservées dans la base de données et configurer le module de collaboration pour laisser passer plus ou moins de données vers les systèmes extérieurs.

6.3 Les travaux futurs

Le projet présenté dans ce document est la première phase du projet globale : retracer la provenance d'une attaque logicielle. Deux autres phases sont nécessaires afin de compléter cet outil. La première est la conception de modules permettant de détecter la présence d'un botnet et la deuxième est la conception de modules permettant de retracer la provenance d'attaques logicielles.

Pour ce qui est des travaux futurs du projet présenté dans ce document, premièrement il serait intéressant de tester l'architecture sur un réseau d'ordinateurs plus performant. Pour ce faire, il serait nécessaire d'utiliser quelques serveurs à capacité de traitement élevée reliée avec des interfaces de communication rapide. Ce qui permettrait d'avoir un

serveur de base de données beaucoup plus performant ainsi qu'un courtier de message plus puissant. Ces tests effectués sur une architecture physique plus performante que celle disponible vont améliorer les performances du système.

Pour finir, l'outil de balancement de la charge pourrait être revu afin de permettre une gestion plus dynamique de la répartition des traitements en offrant des possibilités de configurations plus diversifiées, par exemple en permettant de prioriser des points de sorties.

ANNEXE A

Diagramme de classes du SDK

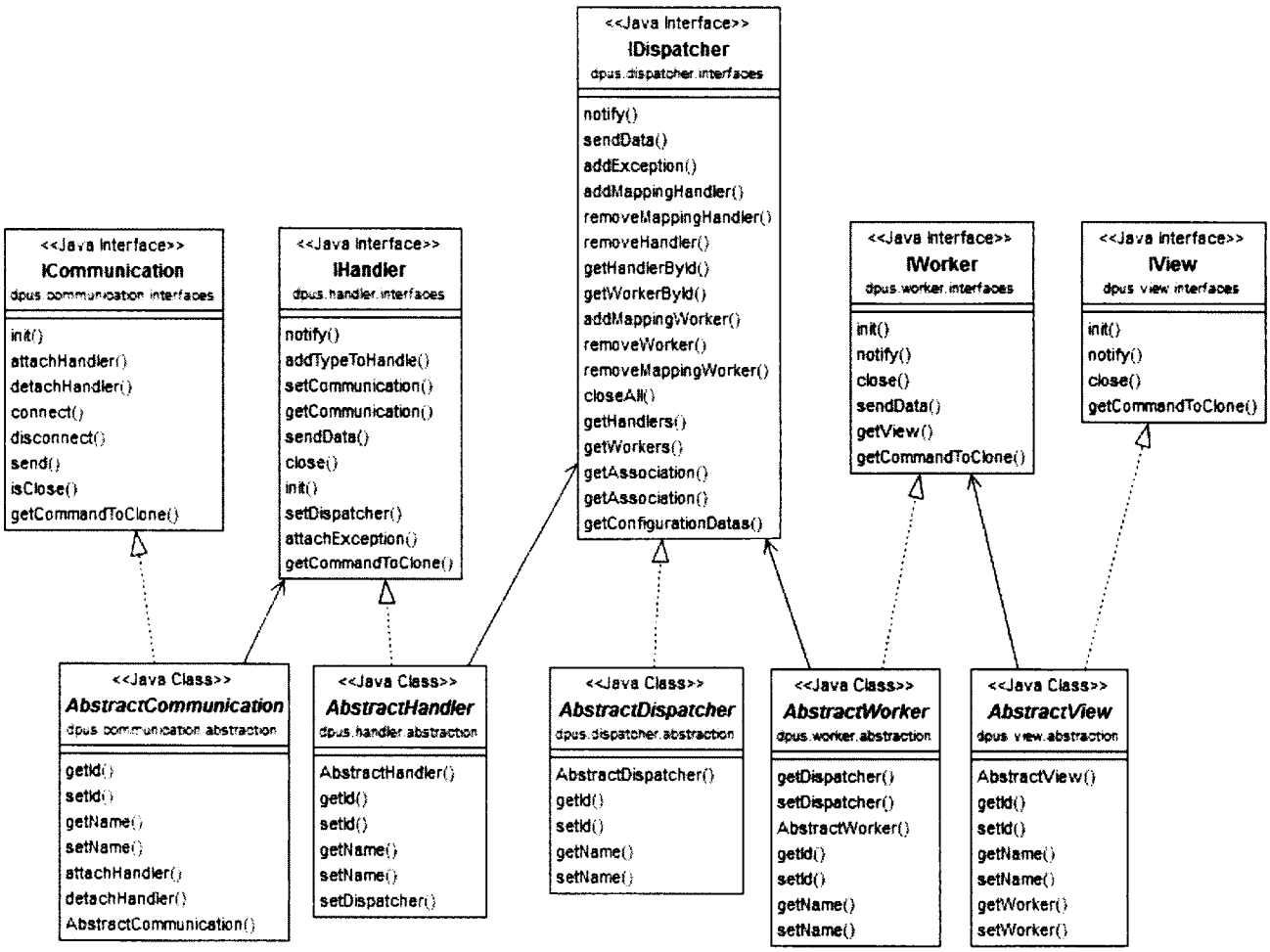

Figure A.1 Diagramme de classe du SDK

LISTE DES RÉFÉRENCES

[1] Abu Rajab, M., Zarfoss, J., Monrose, F. et Terzis, A. (2006). A multifaceted approach to understanding the botnet phenomenon. Dans *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. IMC '06. ACM, New York, NY, USA, p. 41–52.

[2] ActiveMQ (s. d.). *ActiveMQ*. `http://activemq.apache.org/` (page consultée le Mardi le 9 avril 2013).

[3] AMQP (2013). *AMQP*. `http://www.amqp.org/` (page consultée le Mardi le 9 avril 2013).

[4] Apache (s. d.). *Apache Zookeeper*. `http://zookeeper.apache.org/` (page consultée le Lundi le 8 avri 2013).

[5] Apache (2012). *Apache Qpid*. `https://qpid.apache.org/` (page consultée le Mardi le 9 avril 2013).

[6] AsSadhan, B., Moura, J., Lapsley, D., Jones, C. et Strayer, W. (2009). Detecting botnets using command and control traffic. Dans *Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on*. p. 156 –162.

[7] Binkley, J. R. et Massey, B. (2005). Ourmon and network monitoring performance. Dans *2005 USENIX Annual Technical Conference*. USENIX Association, p. 95–108.

[8] BotHunter (s. d.). *BotHunter*. `http://www.bothunter.net/` (page consultée le 8 février 2012).

[9] Dean, J. et Ghemawat, S. (2008). Mapreduce : simplified data processing on large clusters. *Commun. ACM*, volume 51, numéro 1, p. 107–113.

[10] Dingledine, R., Mathewson, N. et Syverson, P. (2004). Tor : The second generation onion router. Dans *Proceedings of the 13th USENIX Security Symposium*. USENIX Association, p. 1–18.

[11] Duda, R. O., Hart, P. E. et Stork, D. G. (2001). Unsupervised learning and clustering. Dans *Pattern Classification*, 2^{e} édition. Wiley-Interscience, p. 526–528.

[12] Feily, M., Shahrestani, A. et Ramadass, S. (2009). A survey of botnet and botnet detection. Dans *Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09. Third International Conference on*. p. 268 –273.

[13] Freeman, E., Freeman, E., Bates, B. et Sierra, K. (2004). *Head First Design Patterns*. O Reilly Associates, Inc.

[14] Glassfish (2013). *Glassfish*. `http://glassfish.java.net/` (page consultée le Mardi le 9 avril 2013).

[15] Gu, G. (2008). *Correlation-based botnet detection in enterprise networks.* Thèse, Georgia Institute of Technology, Atlanta, Georgia, United States of America, 172 p.

[16] Gu, G., Perdisci, R., Zhang, J. et Lee, W. (2008). Botminer : clustering analysis of network traffic for protocol- and structure-independent botnet detection. Dans *Proceedings of the 17th conference on Security symposium.* USENIX Association, Berkeley, CA, USA, p. 139–154.

[17] Hachem, N., Ben Mustapha, Y., Granadillo, G. et Debar, H. (2011). Botnets : Lifecycle and taxonomy. Dans *Network and Information Systems Security (SAR-SSI), 2011 Conference on.* p. 1 –8.

[18] hadoop (2013). *hadoop.* `http://hadoop.apache.org/` (page consultée le Mercredi le 8 mai 2013).

[19] Hibernate (2013). *Hibernate.* `http://www.hibernate.org/` (page consultée le Jeudi le 18 avril 2013).

[20] Java (s. d.). *Java.* `http://www.java.com` (page consultée le 28 Novembre 2012).

[21] Jendrock, E., Ball, J., Carson, D., Evans, I., Fordin, S. et Haase, K. (2010). *The Java EE 5 Tutorial.* `http://docs.oracle.com/javaee/5/tutorial/doc/bncdq.html` (page consultée le 09 Janvier 2013).

[22] Kafka (s. d.). *Apache Kafka.* `http://kafka.apache.org/` (page consultée le Mardi le 9 avril 2013).

[23] Kok, J. et Kurz, B. (2011). Analysis of the botnet ecosystem. *Telecommunication, Media and Internet Techno-Economics (CTTE), 10th Conference of,* p. 1 –10.

[24] Lee, N.-Y. et Chiang, H.-J. (2010). The research of botnet detection and prevention. Dans *Computer Symposium (ICS), 2010 International.* p. 119 –124.

[25] Lin, H.-C., Chen, C.-M. et Tzeng, J.-Y. (2009). Flow based botnet detection. Dans *Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on.* p. 1538 –1541.

[26] Liu, D., Li, Y., Hu, Y. et Liang, Z. (2010). A p2p-botnet detection model and algorithms based on network streams analysis. Dans *Future Information Technology and Management Engineering (FITME), 2010 International Conference on.* volume 1. p. 55 –58.

[27] McAfee Labs (2011). *McAfee Threats Report : Fourth Quarter 2011* (Rapport technique). McAfee, 24 p.

[28] MySQL (2013). *MySQL.* `http://dev.mysql.com/doc/refman/5.7/en/myisam-storage-engine.html` (page consultée le Lundi le 13 mai 2013).

[29] Nogueira, A. a., Salvador, P. et Blessa, F. a. (2010). A botnet detection system based on neural networks. Dans *Digital Telecommunications (ICDT), 2010 Fifth International Conference on.* p. 57 –62.

[30] OCZ (2012). *OCZ Vertex 4 2.5' Solid State Drive.* OCZ Technology Group.

[31] Oracle (s. d.). *Oracle.* `http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136007.html` (page consultée le 10 Janvier 2013).

[32] Oracle (2010). *Java 2 Platform Std. Ed. v1.4.2.* `http://docs.oracle.com/javase/1.4.2/docs/api/java/net/Socket.html` (page consultée le 09 Janvier 2013).

[33] Orocos (2012). *The Orocos Project.* `http://www.orocos.org/` (page consultée le Lundi le 8 avri 2013).

[34] Paxton, N., Ahn, G. et Shehab, M. (2011). Masterblaster : Identifying influential players in botnet transactions. Dans *Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual.* p. 413 –419.

[35] Porras, P. (2009). Directions in network-based security monitoring. *Security Privacy, IEEE*, volume 7, numéro 1, p. 82 –85.

[36] Schiller, C. A., Binkley, J., Evron, G., Bradley, T., Willems, C. et Cross, M. (2007). *Botnets the killer web app.* Syngress, 480 p.

[37] Seagate (2009). *Momentus 7200.4 SATA.* Seagate, rev a édition.

[38] Shvachko, K., Kuang, H., Radia, S. et Chansler, R. (2010). The hadoop distributed file system. Dans *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on.* p. 1–10.

[39] Snort (s. d.). *Snort.* `http://www.snort.org/` (page consultée le 8 février 2012).

[40] Sqalli, M., Firdous, S., Salah, K. et Abu-Amara, M. (2011). Identifying network traffic features suitable for honeynet data analysis. Dans *Electrical and Computer Engineering (CCECE), 2011 24th Canadian Conference on.* p. 001044 –001048.

[41] Storm (2013). *Storm.* `http://storm-project.net/` (page consultée le Lundi 8 avril 2013).

[42] Symantec Cloud Message Labs Intelligence (2011). *March 2011 Intelligence Report* (Rapport technique). Symantec, 11 p.

[43] Symantec Intelligence (2011). *Symantec Intelligence Report : November 2011* (Rapport technique). Symantec, 25 p.

[44] The Honeynet Project (s. d.). *The Honeynet Project.* `http://www.honeynet.org/` (page consultée le 8 février 2012).

[45] Tor (s. d.). *Tor Project.* `https://www.torproject.org/` (page consultée le 8 février 2012).

[46] VMware (2013). *RabbitMQ.* `http://www.rabbitmq.com/` (page consultée le Mardi le 8 avril 2013).

[47] W3School (2012). *w3schools.* `http://www.w3schools.com/browsers/browsers_os.asp` (page consultée le 18 octobre 2012).

[48] Walnes, J. (2012). *XStream.* `http://xstream.codehaus.org/` (page consultée le 14 Janvier 2013).

[49] Wang, H. et Gong, Z. (2009). Collaboration-based botnet detection architecture. Dans *Intelligent Computation Technology and Automation, 2009. ICICTA '09. Second International Conference on.* volume 2. p. 375 –378.

[50] Wang, H. et Gong, Z. (2010). Heterogeneous multi-sensor information fusion model for botnet detection. Dans *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on.* volume 2. p. 428 –431.

[51] Wang, H. et Gong, Z. (2010). Role-based collaborative information collection model for botnet detection. Dans *Collaborative Technologies and Systems (CTS), 2010 International Symposium on.* p. 473 –480.

[52] Wang, H., Hou, J. et Gong, Z. (2011). Botnet detection architecture based on heterogeneous multi-sensor information fusion. *Journal of networks*, volume 6, numéro 12, p. 1655–1661.

[53] Watson, D. et Riden, J. (2008). The honeynet project : Data collection tools, infrastructure, archives and analysis. Dans *Information Security Threats Data Collection and Sharing, 2008. WISTDCS '08. WOMBAT Workshop on.* p. 24 –30.

[54] WebSphere (s. d.). *WebSphere.* `http://www-01.ibm.com/software/ca/en/websphere/` (page consultée le Mardi le 9 avril 2013).

[55] Wu, J., Zhang, L., Liang, J., Qu, S. et Ni, Z. (2010). A comparative study for fast-flux service networks detection. Dans *Networked Computing and Advanced Information Management (NCM), 2010 Sixth International Conference on.* p. 346 –350.

[56] Zhang, X. et Parhi, K. (2006). On the optimum constructions of composite field for the aes algorithm. *Circuits and Systems II : Express Briefs, IEEE Transactions on*, volume 53, numéro 10, p. 1153 –1157.