UNIVERSITÉ PARIS-SUD XI

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ PARIS-SUD XI

préparée au LABORATOIRE DE RECHERCHE EN INFORMATIQUE
dans le cadre de ***l'Ecole Doctorale d'Informatique de Paris-Sud"***

présentée et soutenue publiquement

par

Thomas LARGILLIER

le 29 novembre 2010

Probabilistic algorithms for large scale systems

Directeurs de thèse :
M. Joffroy Beauquier et Sylvain Peyronnet

JURY

Mr. Brian D. DAVISON	Rapporteur
Mr. Aristides GIONIS	Rapporteur
Mr. Serge ABITEBOUL	Examinateur
Mr. Fabio CRESTANI	Examinateur
Mr. Joffroy BEAUQUIER	Directeur de thèse
Mr. Sylvain PEYRONNET	Directeur de thèse

CONTENTS

CHAPTER 1

FRENCH SUMMARY

1.1 Introduction

De nos jours, les systèmes informatiques ont une taille sans cesse croissante pour répondre aux besoins des utilisateurs. Que ce soit dans le domaine du calcul scientifique où de plus en plus d'ordinateurs sont reliés pour répondre à des problèmes sans cesse plus complexes ou dans le domaine du loisir avec un Internet grandissant pour satisfaire toujours plus la curiosité des utilisateurs.

Les défis qui concernent les réseaux à grande échelle sont nombreux: pouvoir garantir aux utilisateurs d'un cluster que leur calcul arrivera à terme et sans erreur dans un temps raisonnable, distribuer des données entre petites entités intelligentes efficacement ou encore protéger le Web contre les tricheurs.

Le but de ma thèse est d'apporter des réponses pour les systèmes à grande échelle afin de garantir l'expérience utilisateur la plus agréable possible.

Les travaux réalisés au sein de cette thèse vont de la conception d'algorithmes à la réalisation de modules pour le noyau d'un système (GNU/Linux). Les domaines étudiés vont des spammeurs sur le Web aux bancs d'essai pour les applications à très grande échelle.

1.1.1 Les tricheurs sur la toile

De nos jours l'Internet est un endroit immense où chaque jour des millions d'internautes font des milliards de requêtes. Afin de les aider, les moteurs de recherches classent les pages sur des critères indépendants de la pertinence à la requête afin de retourner les pages les plus populaires à l'utilisateur.

Le Webspam est un enjeu important économiquement car il permet aux spammeurs de se positionner idéalement sur les requêtes à fort but lucratif. Dès lors il est primordial pour les moteurs de recherche de se débarrasser des tricheurs ou à tout le moins de les reléguer aux dernières pages de résultats.

J'ai travaillé sur la détection et le déclassement du Webspam et dans cette thèse je présente deux méthodes pour lutter contre le Webspam. Ces deux méthodes sont rapides et économes en mémoire, ce qui est primordial lorsque l'on travaille sur un graphe de la taille du Web. J'ai développé ces méthodes avec Sylvain Peyronnet, cela a donné lieu à une publication à la neuvième conférence internationale Web Intelligence en 2010 [61] et à un deuxième article publié au premier workshop international Web Intelligent Systems and Services en 2010 [62]

Les Webspammeurs cherchent à maximiser leur exposition sur la Toile afin d'augmenter leurs revenus. Les moteurs de recherche ne sont pas la seule cible des tricheurs. Avec l'arrivée du Web 2.0, les spammeurs peuvent avoir plus d'influence sur les contenus trouvés sur le Web, notamment au travers des sites collaboratifs.

Les sites d'informations collaboratifs sont des sites Web où les utilisateurs proposent eux-mêmes les liens vers des informations et peuvent voter pour celles qu'ils voudraient voir en première page. Le but des tricheurs est alors d'atteindre la première page pour maximiser leur exposition. Il est donc important de bloquer les tentatives de ces tricheurs afin de rendre nulles leurs attaques sur le système.

Avec Sylvain Peyronnet, nous avons mis au point un schéma de vote robuste pour lutter contre les tricheurs sur les sites sociaux d'information. La méthode s'appelle SpotRank et elle a été utilisée dans un site Web du même nom mis au point par Guillaume Peyronnet. Ce travail a été présenté lors du quatrième Workshop on Information Credibility On the Web [60].

1.1.2 Les systèmes à grande échelle

Le nombre de machines requis pour mener à bien un calcul à tendance à augmenter au fil des années, ceci afin de faire perdurer la loi de Moore. On a donc vu apparaître les clusters puis les grilles de calcul afin de pouvoir mener à bien les calculs scientifiques.

Développer des applications à cette échelle est évidemment plus compliqué lors de la conception mais aussi lors des tests. Il faut être certain de maîtriser un maximum de paramètres afin de pouvoir comprendre et reproduire les conditions expérimentales.

Durant cette thèse, j'ai tout d'abord participé à l'élaboration d'un banc d'essai pour les applications parallèles et distribuées à grande échelle. L'objectif était de fournir un émulateur afin de posséder un contrôle complet de l'environnement pour tester les applications et pas un modèle de celles-ci.

Benjamin Quétier avait réalisé une plateforme où les machines étaient virtualisées grâce à la technologie Xen [5] et où le réseau passait par des machines FreeBSD. J'ai étendu ce banc d'essai en y ajoutant de la virtualisation réseau niveau bas avec le protocole EtherIP [52]. Il s'agit d'un travail joint avec Franck Cappello, Mathieu Jan, Thomas Hérault, Sylvain Peyronnet et Benjamin Quétier ayant donné lieu à une publication à la sixième conférence internationale ACM Computing Frontiers [47] ainsi qu'à un chapitre de livre publié chez IOS Press [46].

J'ai ensuite rejoint un groupe de travail composé d'Aline Carneiro-Viana, Thomas Hérault, Sylvain Peyronnet et Fatiha Zaïdi sur les réseaux de capteurs. Dans ces réseaux qui peuvent être très étendus, les unités de calcul sont très limitées en puissance et en mémoire, de plus leur autonomie est aussi un facteur important car si un trop grand nombre d'entre elles tombent en panne cela peut déconnecter le réseau.

Nous avons proposé ensemble un nouveau protocole de dissémination des données dans les réseaux de capteurs avec collecteur mobile. Notre protocole requiert l'échange d'un nombre de messages exponentiellement plus faible que ce qui pouvait être fait avant. L'article concernant ce protocole sera présenté à la treizième conférence internationale ACM Modeling, Analysis and Simulation of Wireless and Mobile Systems [17].

1.2 Déclassement du Webspam

1.2.1 Le Webspam

Le Webspam désigne l'ensemble des techniques malhonnêtes pour accroître son classement dans les moteurs de recherche. En effet, il est facile mais aisément détectable d'augmenter artificiellement sa pertinence concernant une requête pour un moteur de recherche. De plus les spammeurs cherchent le plus souvent à améliorer le classement de pages pertinentes et intéressantes. En effet sur le Web on ne gagne pas d'argent directement avec les pages de spam.

Les spammeurs concentrent le plus souvent leurs attaques sur les algorithmes de popularité des moteurs de recherche. Ces algorithmes bien connus tel le PageRank [80] de Google où l'algorithme HITS [57] implémenté dans des moteurs anglophones très utilisés comme `http://www.ask.com`. L'algorithme de Google donne un score de popularité *a priori* à toutes les pages. L'algorithme HITS quant à lui classe les pages qui arrivent en résultats à une requête. Dans un cas comme dans l'autre, le calcul de popularité est effectué sur des critères purement structurels et ne fait pas appel au contenu des pages.

Les deux algorithmes se basent sur les liens reçus par les pages web pour évaluer leur popularité (PageRank) ou leur authorité (HITS). La philosophie derrière cette manière de calculer la popularité est la suivante, lorsqu'un site A fait un lien vers un site B, A vote pour B indiquant aux surfeurs qu'ils peuvent y trouver du contenu intéressant qui mérite d'être vu.

Les spammeurs ont très vite compris qu'il était facile d'améliorer artificiellement sa popularité. En effet il suffit de créer soi-même beaucoup de pages web qui vont voter pour la page cible. De ce fait on récupère beaucoup de votes. Ces pages créées dans ce seul but ne reçoivent quant à elles pas de votes donc le leur ne vaut quasiment rien. Toutefois en en créant suffisamment il est possible d'améliorer substantiellement mais frauduleusement son score.

Recevoir beaucoup de votes de petits votants n'est pas une stratégie optimale pour maximiser son score de popularité. Zoltan Gyongyi *et al* ont étudié la structure à mettre en place pour maximiser la popularité d'une page dans le cadre du PageRank dans [38]. Christobald de Kerchove a lui résolu le problème de maximiser le pagerank d'un ensemble de pages. Il propose l'architecture à mettre en place dans [26].

Ces tactiques optimales ont été utilisées par le passé à grande échelle par les spammeurs afin d'améliorer leur score. Toutefois les moteurs de recherche ont rapidement mis en place des contremesures afin de détecter ces comportements

pour ne plus les prendre en compte dans son calcul.

De nos jours, les Webspammeurs mettent en place de nombreuses pages avec une architecture particulière afin de faire circuler le PageRank tout en augmentant le score de la page cible. Ces ensembles de pages créees dans ce but sont communément appelées des fermes de liens.

Il s'agit d'un enjeu important pour les moteurs de recherche. En effet la qualité des résultats est importante afin de gagner et garder la confiance des utilisateurs. Le placement frauduleux des spammeurs sur certaines requêtes nuit à la réputation du moteur de recherche.

1.2.2 Lutter contre le Webspam

On peut diviser en deux parties les techniques de lutte contre le Webspam: la détection et le déclassement. La première méthode cherche a repérer les tricheurs tandis que la deuxième cherche à annuler les effets de la triche sans pour autant pointer qui que ce soit du doigt.

Ntoulas propose une méthode d'identification du Webspam dans [76] basée sur l'analyse du contenu des pages. A partir d'une dizaine de critères il établit un classificateur qui va étiquetter les pages analysées. Cette méthode pose un problème parce qu'elle est coûteuse et les pages promues par le Webspam n'y appartiennent pas. Dès lors il faut identifier tout le Webspam autour d'une page pour réellement faire baisser son pagerank.

Gyongyi *et al* propose dans [39] une méthode pour repérer les tricheurs en diffusant de la confiance dans le graphe à partir d'un ensemble de sites de confiance précédemment sélectionnés. Une méthode similaire est présentée dans [58] par Krishnan *et al*. Elle consiste dans la diffusion d'"Anticonfiance", elle possède les même problèmes que la première méthode à savoir trouver le bon ensemble de départ qui va permettre de couvrir tout le graphe.

Une nouvelle variation de cette méthode est proposée dans [100] par Wu *et al*. La différence c'est qu'ici est pris en compte l'adéquation sémantique.

Une autre méthode consiste à calculer des pageranks biaisés pour ne pas prendre en compte les effets de la triche.

1.2.3 Résultats

La première technique que nous proposons pour combattre le Webspam est basée sur le partitionnement du graphe du Web. En effet les fermes de liens sont très souvent des endroits très denses du Web. De ce fait les tricheurs devraient être

durement touché par un calcul de pagerank où les seuls contributeurs d'une page sont les pages qui appartiennent à un cluster différent.

Cependant les techniques classiques de partitionnement sont trop coûteuses pour être appliquées à l'ensemble du graphe du Web. Nous proposons donc plusieurs méthodes locales pour regrouper les noeuds du graphe.

La première méthode proposée consiste à regrouper les noeuds qui ne font qu'un lien avec la page vers laquelle ils font ce lien. La deuxième méthode consiste à regrouper ensemble tous les noeuds appartenant à des boucles très courtes (3 pas). Enfin la dernière méthode consiste à lancer des marches aléatoires courtes depuis une page et de la regrouper avec celle sur laquelle le plus de marches ce sont arrêtées si ce nombre dépasser un certain seuil.

La première méthode est inefficace tandis que la deuxième et la troisième font baisser significativement le pagerank des pages qui semblent tricher. Toutefois il n'existe de preuve statistique que pour la deuxième méthode de regroupement.

La deuxième technique de lutte contre le Webspam se base, elle, complètement sur les marches aléatoires et ne fait plus de rétrogradation. Les pages profitant du Webspam sont ici identifiées. Pour ce faire on lance une marche aléatoire depuis les noeuds suspects du graphe en ne stockant pas les identifiant des noeuds dans le graphe mais leur distance par rapport au point de départ de la marche aléatoire.

On calcule ensuite un vecteur qui représente la fréquence des n-grammes dans cette marche aléatoire. A partir de ce moment il ne reste plus qu'à identifier des motifs dans ces vecteurs.

Cette technique est plus coûteuse que la première et ne peut être appliquée sur l'ensemble du graphe. C'est pourquoi il faut tout d'abord sélectionner les pages suspectes, *ie* appartenant à un certain intervalle de pagerank ou ayant un degré entrant anormalement élevé.

Il faut aussi posséder une bonne bibliothèque de motifs à comparer avec les vecteurs de fréquence. Nous avons obtenu de très bons résultats en utilisant une petite bibliothèque de motifs.

1.3 Les réseaux de capteurs mobiles

1.3.1 Définition et défis

De nos jours, pour récupérer des données à grande échelle en pleine nature il est courant d'utiliser des sondes disséminées. Ces entités servent à faire des obser-

vations météorologiques, prévenir en cas de départ d'incendie...Ces senseurs sont de petites unités de calcul qui doivent coopérer pour arriver à un résultat. Les données récupérées par les senseurs sont ensuite rassemblées par un noeud collecteur qui ne possède aucune limitation de ressources.

Le noeud collecteur peut être soit statique soit mobile mais doit être relié au réseau par au moins un senseur. Dans le cas où le noeud collecteur est mobile, sa trajectoire au dessus des capteurs mobiles peut être prédéfinie.

Le problème dans les réseaux de capteurs est donc de bien distribuer les données à travers le réseau afin qu'elles puissent être collectées de manière efficace. Il existe 2 approches, l'approche réactive et l'approche proactive. Dans le cas réactif il s'agit d'amener les données vers les collecteurs statiques ou alors de réagir à la position des collecteurs mobiles afin d'amener les données acquises sur sa trajectoire.

Dans le cas proactif, une zone est définie pour le stockage des données dans laquelle le collecteur pourra définir une trajectoire une fois que les noeuds stockant l'information seront connus ou alors se balader aléatoirement dans la zone prédéfinie.

Le plus gros problème à prendre en compte dans les réseaux de capteurs mobiles est le faible niveau de ressources que possèdent les senseurs. Les schémas de distribution doivent être simples et avoir un faible surcoût. De plus les senseurs n'ayant qu'une connaissance locale du réseau, il est important de bien choisir les noeuds qui vont stocker les données.

1.3.2 L'algorithme RaWMS

L'algorithme présenté ci-dessous s'inscrit dans le cadre de la dissémination proactive.

Fonctionnement. Ici il s'agit de distribuer l'information dans un réseau de capteur où l'ensemble du réseau constitue la zone de récupération des données. Chaque noeud possède une mémoire limitée de taille $s(n)$ et ne peut donc stocker l'ensemble des informations du réseau. De plus le collecteur se déplacera librement dans la zone de collecte une fois la dissémination achevée.

Description. Afin de distribuer de manière efficace les données dans l'ensemble du réseau, Bar-Yossef *et al* ont établi une méthode baptisée RaWMS [4]. Dans cette méthode, les noeuds sélectionnent les noeuds à qui ils vont envoyer leur

donnée en utilisant des marches aléatoires. Les marches aléatoires sont légèrement biaisées afin d'arriver à une distribution stationnaire uniforme sur l'ensemble des noeuds.

Chaque noeud répète sa dissémination tous les Δ périodes de temps. De cette manière il est possible de garantir une excellente récupération des données par le collecteur mobile.

Cette approche est efficace car elle permet une bonne récupération des données par un collecteur mobile, de plus elle est très robuste aux pannes des capteurs car les marches aléatoires évitent naturellement les noeuds en panne.

Analyse. Néanmoins cette méthode échange beaucoup de messages puisque pour choisir un noeud la marche aléatoire doit être suffisamment longue. En effet si la distribution stationnaire est uniforme, cela n'est pas forcément le cas après seulement t pas. Il faut donc garantir que t soit supérieur à l'ϵ-temps de mixage de la marche aléatoire. Au delà de ce nombre de pas la distribution obtenue sur les noeuds est ϵ proche de la distribution stationnaire. Le temps de mixage dépend du graphe sous-jacent au réseau mais est dans le cas qui nous intéresse de l'ordre du nombre de noeuds du réseau. De ce fait le nombre de messages à échanger pour garantir une bonne propagation des données du réseau est en $\mathcal{O}(n^2)$.

Dès lors il est important de conserver les points positifs de cette méthode, à savoir la bonne dissémination et la robustesse tout en améliorant le nombre de messages échangés afin de rendre ce dernier plus raisonnable.

1.3.3 Notre proposition

Lors de travaux réalisés en collaboration avec Aline Carneiro-Viana, Thomas Hérault, Sylvain Peyronnet et Fatiha Zaïdi, nous proposons une approche proactive pour la dissémination de données dans les réseaux de capteurs mobiles qui améliorent le précédent résultat d'un facteur exponentiel.

Nous utilisons les mêmes hypothèses que la méthode exposée plus haut, à l'exception de la zone de récupération des données qui chez nous est définie par l'utilisateur et pas forcément égale à l'ensemble du réseau.

L'idée derrière notre méthode est de plaquer une structure sur le réseau afin de guider les marches aléatoires sans pour autant les biaiser.

Notre méthode peut être découpée en trois étapes.

Construction de l'arbre. Afin de garantir un nombre de messages relativement faible pendant la dissémination des données, nous construisons un arbre sur notre réseau. Cet arbre doit être au moins binaire et assurer une distance logarithmique entre toutes les paires de senseurs. Il est primordial de s'assurer que l'arbre n'est pas complètement désiquilibré. Chaque noeud connaît ses fils et son père dans l'arbre.

N'importe quelle technique qui établit un arbre sur un réseau de capteurs peut faire l'affaire comme par exemple PeerNet [28] qui construit un arbre binaire. Lors de nos expériences nous avons utilisé un algorithme glouton, toutefois notre méthode peut être adaptée pour toute topologie garantissant un nombre logarithmique de sauts entre chaque paire de noeuds.

Distribution des poids. Pour choisir la zone dans laquelle les données doivent arriver et quels noeuds doivent être privilégiés, il faut poser des poids sur les noeuds. Un poids > 0 indiquant que le noeud fait partie de la zone de collecte. N'importe quelle distribution marche sachant qu'un noeud i sera au final choisi avec une probabilité $\frac{\text{poids}(i)}{\sum_{k \in \text{Noeuds}} \text{poids}(k)}$.

Chaque noeud fait ensuite remonter son poids dans l'arbre de la manière suivante: les feuilles font remonter leur poids à leur parent, les noeuds internes font remonter la somme de leur poids et de celui de chacun de ses sous-arbres. Au final un noeud stocke son poids plus le poids de chacun de ses sous-arbres, ce qui implique un surcoût en mémoire pour les senseurs. C'est pourquoi il est important que l'arbre reste k-aire avec k assez petit.

Dissémination des données. Chaque noeud fait remonter sa donnée à la racine de l'arbre en un nombre logarithmique d'étapes. Ensuite chaque noeud décide de l'avenir de la donnée dans l'arbre en tirant selon son poids et celui de ses sous-arbres si il conserve la donnée ou dans quel sous-arbre elle doit descendre. Au pire la donnée atteindra une feuille en un nombre logarithmique de pas.

La complexité de notre méthode en nombre de messages est donc la suivante: $\mathcal{O}(n \log(n))$ pour la construction de l'arbre et la propagation des poids. $\mathcal{O}(n \log(n))$ pour la dissémination des données. Ceci représente un gain de $\frac{n}{\log n}$ par rapport à la méthode présentée précédemment.

1.4 Test pour les applications à grande échelle

Les applications développées aujourd'hui le sont à des échelles de plus en plus grandes. En effet la puissance de calcul requise est sans cesse croissante. S'assurer

du bon développement de telles applications est primordial à cause de la complexité de leur mise au point.

Des solutions existent pour tester ces applications. On peut les distinguer en trois catégories. Les simulateurs, les émulateurs et les expériences. Dans un simulateur, l'utilisateur a un contrôle complet sur les conditions expérimentales puisqu'il va simuler toutes les ressources physiques nécessaires. Toutefois à l'interieur d'un simulateur ne tourne qu'un modèle de l'application ce qui ne permet pas de juger l'application finale. A l'opposé se trouve l'expérimentation qui consiste à lancer l'application à tester et voir comment les choses se déroulent. Ici le problème pour les applications parallèles et distribuées est l'allocation de ressources. Il est souvent difficile de pouvoir disposer d'autant de machines que souhaité.

Au milieu de ces deux méthodes, profitant des avantages de l'une et de l'autre, se trouve l'émulation, qui consiste à virtualiser les ressources physiques afin de ne pas être limité et de contrôler l'environnement tout en faisant tourner l'application finale afin de pouvoir récupérer de vraies données sur son exécution.

Des solutions existent, tout d'abord avec des simulateurs tels que Simgrid [18], Gridsim [15], Gangsim [27], Optorsim [7], etc... Mais le problème principal avec les simulateurs reste la précision même si ils réussissent à isoler correctement les processus. Les abstractions faites peuvent cacher des problèmes lors du lancement de l'application et invalider les conclusions du simulateur.

Le nombre de plateformes de test pour les applications distribuées et/ou parallèles à grande échelle a récemment augmenté. Toutefois certaines étant des infrastructures liées à la production comme DEISA [74] ne peuvent être exploitées. Quant à celles dévelloppées pour la recherche, seule Grid'5000 [16] peut correspondre à nos besoins.

En effet, Planetlab [22] connecte les noeuds à travers l'Internet, dès lors il y a un manque de contrôle évident sur les conditions expérimentales puisqu'il est impossible de suivre les paquets de bout en bout. Les résultats obtenus avec Planetlab ne peuvent donc être transposés à d'autres environnements comme montré par [40].

En ce qui concerne Grid'5000 [16], il s'agit de 9 sites géographiques disséminés en France. L'environnement peut être controllé de manière suffisamment fine. Toutefois il lui manque des outils pour tester la robustesse des applications par l'injection de fautes et des outils pour pouvoir sauvegarder les conditions expérimentales afin de pouvoir les rejouer.

Finalement Emulab [50] est une plateforme d'émulation qui offre de la virtualisation à grande échelle ainsi que de l'émulation réseau bas niveau. Le

projet se concentre sur la complète reconfiguration de la pile réseau. Les machines virtuelles utilisées au sein d'Emulab sont basées sur les machines Jail de FreeBSD dont elles sont une extension. Tuer une de ces machines virtuelles revient à tuer un processus ce qui ne permet pas de simuler de vraies pannes machines.

Lors de cette thèse j'ai poursuivi les travaux effectués par Benjamin Quétier sur un émulateur nommé V-ds [82]. V-ds utilise des machines virtuelles Xen pour virtualiser les ressources physiques. Ceci permet de limiter le nombre de machines physiques dont il faut disposer. Les machines communiquent par l'intermédiaire de machines BSD pour assurer une meilleure équité.

J'ai rajouté la virtualisation du réseau bas niveau pour émuler des grilles sur cluster avec l'ajout du protocole EtherIP [52]. Ce protocole permet d'envoyer des trames Ethernet en utilisant le protocole IP. Pour ce faire j'ai utilisé un outil FreeBSD appelé Netgraph qui permet d'intéragir avec le réseau de manière simple et intuitive. J'ai également ajouté un outil qui permet de décrire la topologie que l'on souhaite utiliser en langage DOT[1] pour qu'elle soit mise en place de manière automatique.

1.5 Conclusion

Cette thèse propose des solutions à plusieurs problèmes concernant les systèmes à grande échelle.

Test d'application. Premièrement, avec la poursuite des travaux de Benjamin Quétier *et al* [82], la plateforme V-ds permet maintenant de tester des applications à plus grande échelle en émulant des grilles de calcul sur des clusters grâce à la virtualisation de la couche 2 du réseau avec le protocole EtherIP [52]. Ce travail a été publié en conférence internationale [47] sous forme d'article court ainsi que sous forme de chapitre de livre [46].

Réseaux de capteurs mobiles. Les travaux effectués avec Aline Carneiro-Viana *et al* sur la dissémination de données dans les réseaux de capteurs mobiles ont abouti à un schéma de distribution proactif qui surpasse le précédent schéma d'un facteur exponentiel. De plus notre schéma est plus adaptable dans

[1] `http://www.graphviz.org/doc/info/lang.html`

la mesure où la zone d'arrivée peut être réglée en utilisant un système de poids sur les noeuds et ce sans surcoût pour l'algorithme.

Lutte conte le Webspam. Plusieurs méthodes ont été présentées dans cette thèse pour lutter contre le Webspam. Une pour son déclassement et une pour sa détection. Les méthodes présentées sont légères, ce qui est un caractère indispensable tant le graphe du Web est vaste.

Concernant le déclassement du Webspam, la méthode s'appuie sur le clustering du graphe car il est évident que les tricheurs construisent des structures avec un plus fort taux de clustering pour être sûr que la marche aléatoire du PageRank ne s'éloigne pas trop. Notre méthode a donc choisi plusieurs méthodes de clustering n'utilisant que des informations locales afin de déterminer si cette approche était viable sur le graphe du Web. Il ressort de notre analyse que 2 méthodes arrivent efficacement à déclasser le Webspam avec une preuve statistique pour l'une d'entre elles.

La méthode présentée pour la détection du Webspam utilise les marches aléatoires, outil pratique car proche de la méthode utilisée pour calculer le rang des pages et ne générant qu'un surcoût constant. Lors de la marche aléatoire on stocke la distance des noeuds au point de départ de la marche. Ensuite on regarde la projection statistique de cette marche aléatoire avant de la comparer à une bibliothèque de motifs de triche.

Lutter contre le spam social. En ce qui concerne le spam sur les sites d'information collaboratifs, nous avons présenté une méthode de déclassement du spam nommée SpotRank et un site Web éponyme de cette méthode. L'algorithme est basée sur l'application de filtres statistiques afin de forcer les tricheurs à rentrer dans le rang.

CHAPTER 2

PREAMBLE

Nowadays, computerized systems are at a scale so large that it induces issues regarding the conception, the maintenance, ... Such systems can be divided into two categories, databases indexing a tremendous amount of content and computation infrastructures that gather a huge number of machines (tens of thousands or beyond). Designing algorithms fitted for this scale is a challenge in itself. Designers must realize tractable and efficient algorithms, moreover they should always keep in mind that the applications have to be fault-tolerant to overcome the failures that will happen when using a system this huge.

This thesis consider three kinds of problems,

1. While browsing the Web, one should notice that some content seems to have an inappropriate prevalence. Malicious webmasters try to maximize their visibility. Their manipulations target search engines' ranking algorithm or social websites where the content is produced by the community.

2. Large scale networks like sensor networks have increasing sizes as the cost of sensors becomes cheaper and cheaper. Those networks main actions consists into gathering and diffusing information, it is of the utmost importance to utilize efficient data dissemination protocols that minimize the number of messages exchanged.

3. The development of large-scale applications is fussy, indeed the applications need a good mechanism to distribute the computation and must be fault-tolerant. Even if the algorithm is proved fault-tolerant, it is really important to thoroughly assess the behavior of its implementation in the presence of various adversarial conditions: fail-stop failures or even bizantine behaviors.

2.1 Handling malicious users on the Web

During the early 2000s, the time spent daily on the Internet knew an enormous growth. The online trade knew the same expansion. People realized that it is possible to earn money on the Web without selling any products. By proposing interesting content to users, it is possible to earn massive amounts of money with advertising. This lead to a increasing proportion of users that tried to augment their visibility on the Web using every mean at their disposal.

One of this thesis' objectives is to limit the influence of those users *i.e.*, diminishing their visibility on the Web to propose content fairly exposed to users. This thesis concentrates on malicious users of social news websites and people trying to trick search engines' ranking algorithms by creating many dull pages (so-called Webspam).

2.1.1 Diminishing the influence of malicious users of social networks

With the emergence of Web 2.0[1], users became a more prominent part of the system. It was made easier for surfers to publish content. With the apparition of blogs and wikis, the user generated content is made more visible on the Web.

Users want to produce and promote their own content. The information can travel faster using those new media since today anybody witnessing something important can bring it to the Web via information sites or social networks.

This gave ideas to malicious users, since being on the first page of a site like digg[2] may redirect a huge amount of traffic to your site. Their objective is to manipulate the algorithm that rank news by manipulating the votes for their news in order to appear closer to the top position and secure a place on the front

[1] `http://oreilly.com/web2/archive/what-is-web-20.html`
[2] `http://digg.com`

page. These users may regroup into "cabals" (collusion of users) in order to maximize the efficiency of their manipulations.

In chapter 4, I describe a robust voting scheme that prevent spam content from reaching the front page.

2.1.2 Reducing the boosting effect of Webspam

People quickly understood that it is possible to earn money on the web without selling anything, only through advertising's income. Thus they need to attract as many users as possible on their pages. Since a huge portion of the traffic comes from search engines, the higher you appear on the result's list the bigger the chance is that users will click on the link.

Search engines have many mechanisms to sort pages that appear on the results list, the two principal are the relevance and the popularity mechanisms. The easier to manipulate is the relevance one. But search engines' countermeasures regarding these manipulations are now really effective. Moreover the pages promoted are usually genuine pages that do not need pertinence boost. The metric often abused by malicious users is the popularity one. It is the PageRank in the Google search engine.

One way of artificially increasing the popularity of one page p is to create many pages that will make one outlink towards the target page p. This technique is deprecated and easy to spot. Cheaters indulge in a weapon race with search engine regarding the promotion of their pages. They now organise their dull pages into specific structures to maximize the obtained PageRank.

Chapters 5 and 6 present techniques that respectively demote Webspam and detect pages benefiting from Webspam.

2.2 Large scale networks

Networks have increasing sizes introducing scalability issues into the development of applications supposed to run on them. People tend to use more and more computers for a single computation to solve bigger and bigger instances of problems. The cost of one unit of a network (either a sensor or a computer) tends to decrease, therefore it is possible to use bigger systems reaching sizes in tens of thousands or beyond.

Applications developed for such systems really need to decrease the amount of communication between units. Those applications should also be fault-tolerant.

Since they use a huge number of machines, the probability that at least one will fail during the experiment or the computation is high on the long-run. Therefore users need to thoroughly test their application during the development.

2.2.1 Improving data dissemination in sensor networks

Sensor networks are principally used to monitor a zone. They can be used to detect abnormalities and send a early warning for natural catastrophes. In order to do so, sensors collect and report data to decision entities. The size of these networks increase as the price of a single sensor decrease.

In our model of those networks, sensors are really simple machines with limited resources. They don't have any indications regarding the topology of the network, each node only knows its neighbors. Nodes' data is collected by an external entity that may be either static (the data should be brought to it) or mobile (travel over the network to collect it).

When the collecting entity is mobile it is important to ensure that it does not need to cover the whole network to gather all the information. Then the network should possess a decent data dissemination scheme to ensure that the data is well distributed over the recuperation zone.

I present in chapter 8, a new data dissemination scheme that can be used in wireless sensor networks to report the data. It requires a number of exchanged messages much smaller than previous approaches.

2.2.2 Realistic evaluation of parallel and distributed applications

Developers of large scale applications, for clusters and grids for example, have to face many issues while developing. They must ensure that, while attaining high performances in terms of computation steps, the computation must end correctly. It is well known that messages will be lost or altered and machines will crash at this scale.

There exists several ways to ensure that an application will be fault-tolerant. Developers may use a proved algorithm, verify the model of their applications,... But in the end it is of the utmost importance to test the written application since some bugs may have slipped during the development. The testing phase must be as extensive as possible to guaranty that the application will behave well in the presence of a maximum of adversarial conditions.

I present in chapter 9 a testbed for large scale systems that authorizes the user to test his application against failures at both machines and network level.

Part I

Fighting malicious behaviours on the Web

CHAPTER 3

INTRODUCTION

The World Wide Web [9] represents a huge data collection organised in Web pages. The Web pages are accessed using the HyperText Transfer Protocol [30] (HTTP) and connected through hyperlinks. The navigation inside this collection is made almost exclusively through search engines.

At the beginning of the 2000s, the commercial Web arises. There are two ways of making money on the Web: the first is through a commercial website where you sell products online. It is also possible to earn money without anything to sell, only by displaying ads on your websites. You can be rewarded either after a sufficient number of display of the ads or at each click made on the ad by a visitor.

On the Internet the majority of earnings are made with advertisement, for regular websites as well as for search engines. This could be explained by Google's strategy that bases its income mostly on Adwords. Adwords is an application where sellers bid on search words so their publicity will appear when those words are typed in Google if they are amongst the highest bidders. Everybody wants to maximize their profit, and it is basically proportionnal to their number of visitors. Genuine websites need to increase their visibility on the Web while search engines need to secure the loyalty of their users.

In order to do so, they need to provide users with interesting and popular websites. Therefore they need to make sure no cheater makes his way up to the

top of their ranking using malicious techniques.

With the advent of Web 2.0 [79], Internet users got more control on the accessible content on the Web. It has eased the cheaters becoming more visible on the Web since they could "steal" links in blogs comments for example.

Malicious users tend to apply several techniques to maximize their visibility, the first one being the creation of dull pages which only objective is to create popularity that will be directed to a specific target page. This manipulation is referred to as Webspam. It is important to cheaters that this pages are indexed by search engines, that's why they "steal" links over the Internet.

Another way to gather traffic is through collaborative websites, where users propose content and vote for content they like the most. All the content is then sorted according to the votes and the top news are displayed on the front page of the website.

I designed with Sylvain Peyronnet a robust voting scheme for social news websites called SpotRank. This scheme voids the effects of the attacks made by cheaters on collaborative news websites. It was implemented in a website[1] developed by Guillaume Peyronnet. SpotRank was presented at the 4th Workshop on Information Credibility on the Web (WICOW 2010) [60] in conjunction with 19th World Wide Web Conference (WWW 2010).

Fighting Webspam is a weapon race between search engines and cheaters, each side constantly re-enforcing its arsenal. Techniques to detect or demote the effects of Webspam have to have a low computational cost because the graph is huge and evolve to remain efficient overt time. In chapter 5, I describe a method to demote the effects of Webspam using lightweight clustering techniques. This work is published in the proceedings of the ninth international Conference on Web Intelligence (WI 2010) [61]. This work is co-authored with Sylvain Peyronnet.

I also present in chapter 6 another method that detects pages benefiting from Webspam using random walks from suspected nodes. This work is again realised with Sylvain Peyronnet and will be published in the proceedings of the 1st International Symposium on Web Intelligent Systems & Services (WISS 2010) [62] with the 11th International Conference on Web Information System Engineering (WISE 2010).

The following introduction is organised as follows. In section 3.1 the functioning of collaborative news websites and their weaknesses to cheaters attacks are presented. Studies and ranking methods for social websites are introduced in

[1] `http://www.spotrank.fr`

section 3.2. Section 3.3 focuses on Webspam. In section 3.4 existing methods to fight Webspam are presented. Last, sections 3.5 and 3.6 introduce some of the tools I used to fight Webspam and social spam.

3.1 Collaborative news websites

In the last years, the way people interact with each others on the Web has drastically changed. Websites now provide information which is an aggregation of user-generated content, generally filtered using social recommendation methods to suggest relevant documents to users. The most known example of such a website is *Digg*[2]. This is a social news website: people share content they found on the web through the Digg interface, then users can vote for the news they like the most. Voting for a news is then considered as a recommendation, and (according to the result of a non disclosed algorithm) news with a sufficient number of recommendations are displayed on Digg's front page.

Digg has been launched in November 2004, and since then numerous Digg clones (generally denoted as *Digg-like*) were created by webmasters. This huge success can be explained by the amount of traffic such a website aggregates and redistributes. Indeed, being on the front page of a website such as Digg seems to be very interesting since repeated testimonies amongst webmasters state that thousands of unique visitors are obtained within one day for a website on the front page of Digg (or similar sites). Since most websites follow an economic model based on advertisement, obtaining unique visitors is the best way to improve the income. It is then tempting for a user to use malicious techniques in order to obtain a good visibility for his websites.

A malicious technique is explained in details by Lerman in [64] where she recalls the Digg 2006 controversy. This controversy arose when a user posted on Digg an analysis proving that the top 30 users of Digg were responsible for a disproportionate fraction of the front page (latter studies ensure that 56% of the front page belong to the top 100 users only). This means that the top users are acting together in order to have their stories (e.g websites they support) displayed on the front page. The controversy led to a modification of the Digg algorithm in order to lower the power of this so-called *bloc voting* (collusion between a subset of users).

Since 2006, malicious users became more and more efficient (see for instance the paper of Heymann *et al.* [49]). Cabals (collusion of large group of users that

[2] `http://digg.com`

vote for each others) have been automatized using daily mailing lists, some users post hundreds of links in order to flood the system, others have several accounts and thus can vote for themselves (using several IP addresses) etc. To the best of our knowledge, no social news website implements a robust voting scheme that avoids the problem of dealing with malicious users while still providing a high quality of service (i.e. providing relevant news to users).

3.2 Fighting social spam

Regarding the analysis of social news websites there are only a few research papers available. The work done by Lerman (and her coauthors) in many papers [64, 66, 65, 81, 63] is probably the most extensive done in this field. These papers analyse the behavior of users and content in social sites such as Digg and *Flickr*[3]. Abstract modeling of users is done and allows to infer the dynamics of users' rank [63], but also to predict which news can obtain good ranking according to the first votes [66]. However, this work is analytic, the goal is to understand how social news websites work. We, on the other hand, aim at designing a robust voting scheme in an adversarial environment, thus our approach is normative.

In their paper [49], Heymann *et al.* present a survey on spam countermeasures for social websites. They sorted three categories of such countermeasures: identification-based methods (*i.e.* detection of spam and spammers), ranked-based method (*i.e.* demotion of spam) and limit-based method (preventing spam by making spam content difficult to publish). Clearly SpotRank falls in the scope of ranked-based method since our goal is to reduce the prominence of content that benefit from malicious votes.

Bian *et al.* [11] describe a machine learning based ranking framework for social media that is robust to some common forms of vote spam attacks. Some other work focusing on manipulation-resistant system, and using a notion close to the one of pertinence, can be found in [85].

A related field of research is the detection of click fraud in the Pay Per Click (PPC) advertising market, but also in Web search ranking. In PPC, webmasters display clickable advertisements on their website and are paid for each click going through the ad. In Web search ranking, the more a link to a website is used, the higher the site is ranked. For instance Jansen [55] give details of the impact of malicious clicks in PPC while Metwally *et al.* [72], Immorlica *et al.*

[3] `http://www.flickr.com/`

[53] give strong analysis of the phenomenon together with algorithms to cope with it. Radlinski and Joachims [83] focus on randomized robust techniques that infer preferences from click-through logs.

The problem of providing the users of a community with a good selection of news seems to be a recommendation problem. Cosley *et al.*[24] study the relation between recommendation systems and users, while Lam and Riedl [59], and O'Mahony *et al.* [78] address the problem of malicious users and robustness of systems. However, recommendations by friends has been proven to always be better than recommendations using automatic systems (see for instance the papers of Sinha and Swearingen[92]). To overcome this problem, researchers from the recommendation systems field introduce the notion of trust as a reflection of users' similarity.

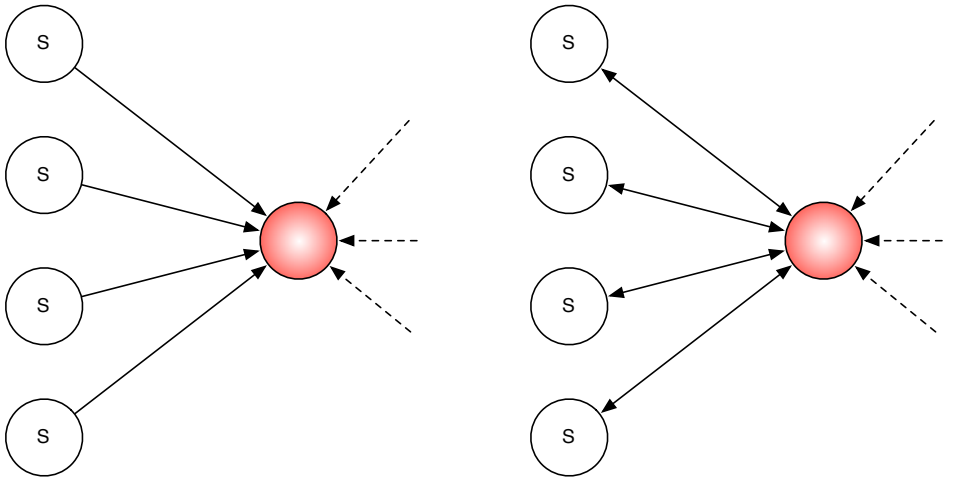
In this thesis, I focus on completely different techniques that demote votes that are malicious, or done by users known to be malicious. The approach I developed together with Sylvain Peyronnet does not use machine learning methods and is based on the notion of *pertinence*. It is worth noting that despite the lack of research papers in this field, there are probably a lot of undisclosed work going on in social news websites' teams.

3.3 WebSpam presentation

Notions and techniques introduced in the second part of this section are from [38] and [26].

Search engines rank Web pages according to two kind of metrics while delivering results to the user. Relevance metrics and popularity metrics. The relevance score depends on the requests made by the user and is self-contained within each Web page. The popularity mechanism is on the other hand content independent and is most of time structurally related. An hyperlink from page p_1 to page p_2 means in a human context, that webmaster of page p_1 recommends page p_2. It is similar to a vote and many popularity mechanisms rely on hyperlinks to determine the popularity of one Web page.

Since the algorithm behind each search engine is broadly known, many users who want to maximize their exposition try to maximize their score. This lead to a new field of research called Search Engine Optimization (SEO). SEO consists of all fair techniques, *ie.* approved by search engines, to improve the ranking of a website or of a particular Web page. It tries to acquire "votes" through fair referencing and doesn't attempt to play the algorithm.



(a) Simplest example of spamdexing where Webspam is marked with an S and the target page is in red.

(b) Optimal structure to maximize a unique page's pagerank

Figure 3.1: Structures to increase the pagerank of one target page.

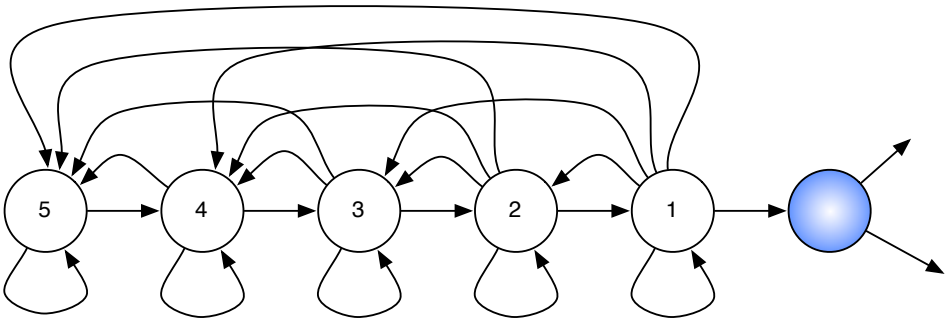


Figure 3.2: Optimal structure to maximize the pagerank of a set of 5 pages.

Other webmasters are ready to use whatever technique that helps increasing the ranking of his Web pages. Techniques not considered SEO are often referred to as spamdexing. Spamdexing can take many forms. The first one and the easiest is to try to trick the algorithm on the relevance of a specific Web page regarding a certain request. Multiplying the occurrences of keywords, providing a different page for crawlers and users, ... These techniques are nowadays well-known by search engines and do not affect anymore the ranking of one page.

Then malicious webmasters try to increase their popularity instead of their

relevance. It is worth noting that profitable pages are genuine pages and not forged Webspam. There are three kinds of pages on the Web for a webmaster. *Inaccessible* pages that she can not alter in any way, *accessible* pages which can be slightly modify by her like forums or blogs through their comments and *own* pages that belongs to the webmaster and that she shapes like she wants. The objective of the webspammer is to steal a maximum number of links from accessible pages in order to index her Webspam in search engines. They may use several techniques like creating *honey pots*, technical pages like Unix documentation for example, that are naturally linked by users. They can also infiltrate web directories, steal links from blogs' comments, forums or wikis. They also participate in link exchange between spammers or buy domains when they expire to hijack links made to these domains.

Indexing the Webspam is just one step required to artificially increase the pagerank of the target page(s). The Webspam has to be organised into a specific structure to maximize its efficiency.

In order to increase the popularity score of one particular page, cheaters create many dull pages that will be organised in a specific architecture to maximize the popularity while avoiding automatic detection. Fig 3.1a represents the simplest architecture a cheater can put in place to increase its score. He creates as many spam pages as possible and all these pages only make one hyperlink to the target page. Dashed lines represent stolen links from accessible pages.

This structure is easily identifiable and easy to counter. It is deprecated nowadays and can not be employed at large scale. Cheaters make these structures evolve constantly. Once one has been discovered and made inefficient by search engines, it is discarded and slightly modified to pursue its objective.

In [38], Gyongyi *et al* propose the optimal structure to increase the pagerank of a single page. This structure is presented in Fig 3.1b. It is very similar to the structure in 3.1a, the difference being that links are mutual. Indeed spammers realized that the PageRank needs to circulate in order to grow.

De Kerchove in [26] solves the problem of maximizing the pagerank of a set of pages. The optimal structure for k pages numeroted from 1 to k is as follows,

- $\forall 1 < i \leq k$, i links to $i - 1$,
- $\forall i \in [1, k], \forall i \leq j \leq k$ i links to j,
- 1 links to a page outside of the set (represented in blue in Fig 3.2).

This structure is represented for a set of 5 pages in Fig 3.2. To maximize the pagerank of several pages, De Kerchove maximized the time's expectancy the

random surfer will spend on these pages thus leading to an optimal structure. Both papers are exhibiting optimal structures for this purpose. However, since these structures have a very rigid architecture (thus being easy to detect) , spammers are constructing less efficient, but slightly different structures. As we will see in the next sections, our method performs well even on slightly modified spamming structures.

3.4 Fighting Webspam

Since search engines are based on well known popularity metrics such as the PageRank [80], spammers are trying to artificially boost their websites with respect to these metrics. At the same time, Webspam has been extensively studied since it is really important for search engines to be able to deal with it.

More recently the interest was concentrated on the evolution of Webspam and Chung *et al.* proposed a link spam study in [23] to see how it evolves through a series of Web snapshots.

With the apparition of Webspam, many techniques were developed to deal with its effects in order to ensure the user with a fair ranking. There are two basic types of methods to deal with Webspammers: the first one is detection, *i.e.* identifying Webspam or pages benefiting from it and demotion whose objective is to void the effects of such techniques without explicitly identifying Webspam.

In the spirit of the first category, Ntoulas *et al.* propose to identify spam through content analysis. The method use several criteria presented in [76]. Unfortunately, this method does not scale up since the Web is growing too fast and studying every page in depth is way to costly. Other approaches to identify cheaters on the Web rely on the structure of the Web and not on the content of the web pages.

The paper [37] by Gyongyi *et al.* proposes a method whose goal is to identify link spam through the estimation of pagerank coming from spam pages for every node. It is first required to find a set of "good" pages and then run a biased PageRank to find the part of all pages' pagerank that come from "good" pages. The drawback of this method is that it requires a preprocessing human step where people label pages as spam or non-spam.

Benczur *et al.* (see their paper [8]) propose a fully automatic detection method for Webspam by observing the distribution of contributing pages to suspected pages. Those who appear to have a biased distribution are considered as spam by their method.

Authors of [39] (resp. [58]) propose to give a Trust (resp. AntiTrust) score to pages to fight Webspam. These methods oblige the user to find a set of pages she trusts (resp. distrusts) and then use a propagation scheme *a la* PageRank. It is then tricky to find a good seed set that will cover the whole graph. The notion of TrustRank was then refined by Baoning *et al.* to add topicality in [100]. Authors of this paper first partition the seed set into topically coherent groups before computing the TrustRank for each topic. The final Trust score of one page is a combination of its topic-specific Trust scores. This approach improves the result obtained with TrustRank regarding the demotion of Webspam.

Andersen *et al.* in [2] propose to compute a Robust PageRank by first approximating the supporting set of each page *i.e.* the set of pages that contribute to its pagerank.

3.5 Random walks

A random walk on a graph consists in a simple algorithm that travels the graph. The next step of the walk is chosen uniformly at random between the neighbors of the current node. When the graph is finite, it is equivalent to a Markov Chain. A Markov chain is a random process where both time and space are discrete. Markov chains notions presented in this section are from [13, 75].

Let V be a countable set representing the nodes (called states) n_i of the graph.

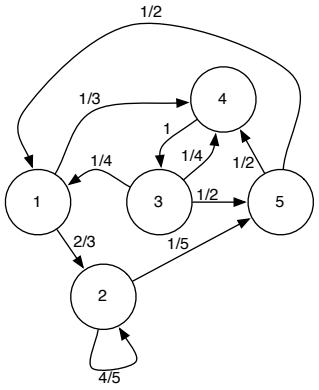
Definition 1 *Let $(X_n)_{n\geq o}$ a series of random variables valued in V. $(X_n)_{n\geq o}$ is a Markov chain if*

$$\mathbb{P}(X_n = j | X_o = i_0, \ldots, X_{n-2} = i_{n-2}, X_{n-1} = i) = \mathbb{P}(X_n = j | X_{n-1} = i)$$

for all $n > 0$ and all n-tuple of states $i_0, \ldots, i_{n-2}, i, j$ for which both sides of the equality are defined.

A Markov chain is a series of random variables where the value of one variable depends only on the value of the one before and not on the rest of the history. A Markov chain is said to be *homogeneous* if $\forall n, \mathbb{P}(X_n = j | X_{n-1} = i)$ depends on i and j and not n. In this case, it is possible to define the *transition matrix* $P = (p_{ij})_{i,j\in V}$ associated to the Markov chain where $p_{ij} = \mathbb{P}(X_n = j | X_{n-1} = i)$. In the following we always consider homogeneous Markov chains.

The transition matrix P may be represented by its *transition graph* G whose nodes are states in V. In G there is an arc $i \to j$ if and only if $p_{ij} > 0$. In this



$$P = \begin{pmatrix} 0 & 2/3 & 0 & 1/3 & 0 \\ 0 & 4/5 & 0 & 0 & 1/5 \\ 1/4 & 0 & 0 & 1/4 & 1/2 \\ 0 & 0 & 1 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \end{pmatrix}$$

Figure 3.3: Markov chain and its associated transition matrix.

case, the arc is labeled by p_{ij}. Fig 3.3 represents a Markov chain together with its matrix. It is interesting to notice that Markov transition matrix are stochastic, *i.e.* $\sum_j p_{ij} = 1$.

It is interesting to study the probability (noted $p_{ij}^{(n)}$) of going from i to j in n steps in a Markov chain.

$$p_{ij}^{(n)} = \mathbb{P}(X_n = j | X_0 = i) = \mathbb{P}(X_{n+k} = j | X_k = i) \quad (n \geq 1, k \geq 1)$$

The n-steps transition matrix is noted $P^{(n)}$ and it can be shown that $P^{(n)} = P^n$.

Let's now introduce the *state probabilities*: $\pi_k(n) = \mathbb{P}(X_n = k)$. The distribution of X_n can be written as vector with $\pi(n) = (\pi_1(n), \pi_2(n), \dots)$ where $\sum_k \pi_k(n) = 1$. We then have,

$$\begin{cases} \pi(n+1) = \pi(n)P \\ \pi(n) = \pi(0)P^n \quad \forall n \geq 0 \end{cases}$$

The results above allow us to say that a Markov chain is completely defined if both its transition matrix and the distribution of X_0 are known. While speaking of the probabilities $\pi_k(n) = \mathbb{P}(X_n = k)$ of a Markov chain regarding the number n of steps, one is studying the *transient state* of the random process. Generally, the distribution of X_n depends on the time and the initial distribution $\pi(0)$. If the distribution $\pi(n)$ converges, as n tends to the infinite, towards a limit distribution noted π, then π is the steady-state of the stochastic process and does not depends on the initial distribution $\pi(0)$.

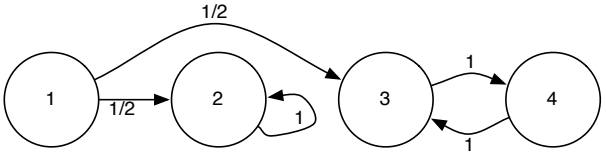


Figure 3.4: Example of a Markov chain.

If $\lim_{n\to\infty}(\pi(n)) = \pi$ exists, independently from the initial distribution and if π is a probability distribution, we say that the Markov chain $(X_n)_{n\geq 0}$ converges towards π or has a limit distribution π.

A discrete probability distribution π is *stationary* regarding the stochastic matrix P if $\pi P = \pi$. A Markov chain is stationary if the distribution $\pi(n)$ of the random variable X_n does not depend on n, *i.e* $\pi(0)$ is a stationary distribution of the random process. The following theorem concerns the existence of such stationary distributions.

Theorem 1 *For a finite Markov chain, there always exists at least one stationary distribution. This is not the case when the state's space is infinite.*

To formulate the uniqueness criterion, we must first introduce a classification on the states of a Markov chain.

Two states are *communicating* if we can go from i to j and from j to i, *i.e* $\exists m, n \in \mathbb{N}, \quad p_{ij}^{(m)} > 0$ and $p_{ji}^{(n)} > 0$. It is then possible to define an equivalence relation on the set V and then we partition V in disjoint classes, $V = C_1 \cup C_2 \cup \cdots \cup C_r$. Any pair of states from the same class are always communicating while states in different classes never communicate. It is then possible to distinguish two kinds of classes,

- A class is *transient* if it is possible to go outside of the class, but in this case the process can never re-enter this class.

- A class is *recurrent* if it is impossible to go outside of the class.

The uniqueness criterion is in the following theorem,

Theorem 2 *A finite Markov chain has a unique stationary distribution if and only if it has only one recurrent class.*

In Fig 3.4, $\{1\}$ is a transient class while $\{2\}$ and $\{3,4\}$ are recurrent classes. Since there is strictly more than one recurrent class, there is not an unique stationary distribution. The computation gives that $xP = x$ if $x = (0, 1-2\alpha, \alpha, \alpha)$ with $\alpha \in [0, 1/2]$.

If the unique recurrent class of a Markov chain $C = V$, *i.e* all states are communicating, the Markov chain is said to be *irreducible*.

Random walks are part of the computation of the PageRank of Web pages. They mimic the behavior of the random surfer starting from a random page and following hyperlinks to continue surfing. To calculate the PageRank, the random surfer can at each step, either choose an hyperlink uniformly at random from all present on the current page with probability p or with probability $1-p$ choose any page from the index uniformly at random. This prevent the random surfer to be stuck on one page without any outlink.

My hypothesis is that random walks can provide useful structural information for Webspam demotion or detection. Our first approach was not precise enough, it consisted in identifying link farms using random walks with a limited storage capacity, *i.e.* random walks that remember their k last steps. Using those traces we looked at the frequency of appearance of each node in the memory, *i.e.* the number of times a node was visited during the walk. Depending on the number of nodes found with a high frequency in the random walk, we tried to dissociate good nodes from spam ones. A more elaborate version, where random walks do not store nodes' id but their distance to the starting point of the walk is presented in chapter 6 that proves the validity of the hypothesis.

3.6 Clustering

The objective of clustering on a graph is to regroup nodes that are densely connected, *ie* nodes between which exists many paths. This corresponds to groups of pages on the Web. Fig 3.5 represents the execution of a clustering algorithm on a small graph where clusters are identified through ids and colors.

There are many ways to regroup nodes and graph clustering was extensively studied [90]. Two widely used techniques are the Markov Clustering Technique [94] and the Edge Betweenness Clustering [35]. I present both techniques in the remaining of this section and explain why while efficient on smaller graphs they are of no use on the Web graph.

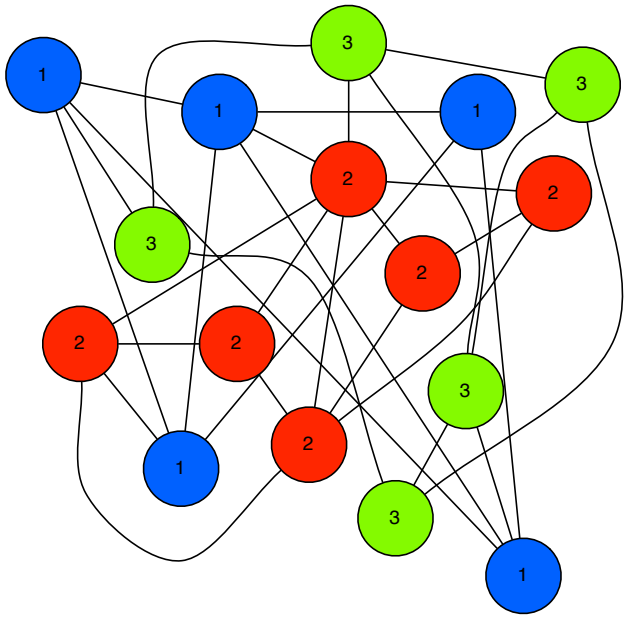


Figure 3.5: Clustered graph.

3.6.1 Edge betweenness centrality

This approach proposed by Girvan and Newman in 2003 [35] is based on the "edge betweenness" which is an extension of the "node betweenness" centrality notion. This notion first appeared in the field of social interactions [98] to determine the role of actors (nodes) inside a network (graph), the objective being to identify communities of sharing interest. This notion is based on the shortest paths in a graph. Between any pair of nodes i,j it may exists several paths from i to j. Some of them may be paths of shortest length (in number of nodes on the path). The intuition is to regroup "central points" in the graph to regroup nodes.

The "betweenness centrality" between i and j of node k is the number of shortest paths from i to j passing through k (noted $\sigma_{ij}(k)$) divided by the number of shortest paths form i to j (σ_{ij}). Then the "node betweenness centrality" of node k is the sum of its betweenness centrality for all possible pair of nodes.

$$C_B(k) = \sum_{i \neq j} \frac{\sigma_{ij}(k)}{\sigma_{ij}}$$

where $i \neq k$ and $j \neq k$.

Girvan and Newman extended this notion to edges. The "edge betweenness centrality" of edge $\{k, k'\}$ is the following sum over all pair of nodes i, j different from both k and k'

$$C_B(\{k, k'\}) = \sum_{i \neq j} \frac{\sigma_{ij}(\{k, k'\})}{\sigma_{ij}}$$

where $\sigma_{ij}(\{k, k'\}$ represents the number of shortest paths from i to j that contain the edge $\{k, k'\}$.

Considering a graph with two clusters, sets of nodes densely connected, with few connections between clusters. Shortest paths between nodes in different clusters must take the few edges connecting the clusters. Those edges are then provided with a high "edge betweenness centrality". The EBC method is based on this idea. By iteratively removing edges with decreasing centrality, clusters will be disconnected. The method to discover clusters is an algorithm in two phases

1. *Betweenness computation*
   Compute the edge betweenness centrality of all edges. Remove the one with the maximal score and recompute the centrality for all edges until they are no more edges in the graph.

2. *Cluster construction*
   At the beginning every node is its own cluster. Arcs between clusters are added in the inverse order from which they were removed in the previous phase. If an arc joins two nodes in different clusters they merge.

The second phase produces iteratively sets of clusters on the graph. By adding arcs in the inverse order from which they were removed, arcs added first must be intra-clusters arcs while arcs added in the end are inter-clusters arcs. It is important to determine the moment the second phase stops adding intra-clusters arcs and begins connecting clusters. Girvan and Newman propose a metric called "modularity",

$$\mathcal{M}(C) = \sum_{c=0}^{N_C} \left[\frac{d_c}{L} - \left(\frac{l_c}{L} \right)^2 \right]$$

where N_C is the number of clusters in a set C, d_c is the number of intra-cluster arcs in a given cluster c, l_c is the total degree (number of arcs) of nodes in the cluster c and L the total number of arcs in the graph. The modularity is computed on the raw graph before any arc removal. This metric is equal to 1 if all arcs in

C are intra-cluster arcs and 0 if the proportion of intra- and inter-cluster arcs is equal. The optimal set of clusters is thus the one with the maximal modularity in the second phase of the algorithm.

This method belongs to the ones that scale up best reaching $100,000$ arcs graphs. We are far from the size of the Web graph and thus this method won't scale up to our needs since it is impossible to compute the "edge betweenness centrality" for every edge of the graph and the modularity in the second phase. Those operations require a global knowledge of the graph and have a too high complexity to be computed as often as required by the fight against Webspam.

3.6.2 Markov clustering technique

The MCL method simulates stochastic flows to partition the graph into clusters. It is a fast approach that is only limited by the number of nodes in the graph. It was designed by Stijn van Dongen [94].

As said previously clusters are groups of nodes tightly connected. Thus there should exist many paths of bigger length between nodes that belong to the same cluster. The number of these paths is expected to be bigger that the one between two nodes in different clusters. From another perspective, random walks in a graph have a few chances to leave a cluster. The intuition behind the algorithm is the same as the human behavior, find huge "blocks" densely connected with few connections between blocks.

The MCL method finds the cluster structure of a graph by using a bootstrapping technique. The process deterministically computes the probabilities of the random walks in the graph, using two operators transforming a set of probabilities into an other. This is made possible by the use of column stochastic (or Markov) matrices (see previous section).

The MCL algorithm simulates random walks on the graph by alternating two operations: the expansion and the inflation, both parameterized. The expansion operator consists in taking the stochastic matrix to the power k using classic matrix multiplication. The inflation operator Γ takes the matrix to its r^{th} Hadamard power with a normalization step afterward to keep the matrix stochastic *i.e* cells of the matrix are probability values.

A column stochastic matrix is a non-negative matrix whose columns sum to 1. Considering a column stochastic matrix M and a real $r > 1$, the result of the

application of the inflation operator is the following,

$$\Gamma_r(M_{ij}) = \frac{(M_{ij})^r}{\sum_i (M_{ij})^r}$$

With a coefficient $r > 1$ it is easy to see that the application of the inflation operator will favor more probable walks for a particular node (column of the matrix).

The expansion phase computes the probabilities for random walks of longer length thus changing the values of the matrix. The value M_{ij} being the probability that a random walk starting in j stops in i after a given number of steps depending on the parameter k of the expansion. Since nodes in clusters are densely connected, the probability will be much higher that a random walk starts and stops in the same cluster that it takes an inter-cluster arc. Thus the inflation operator will increase the probability of intra-cluster paths and lower the probability of random walks using inter-cluster arcs. This is done without any prior knowledge about the organisation of the graph in clusters but simply due to the presence of clusters in the graph.

Finally iterating the expansion and the inflation operators will separate the graph into groups of nodes that will be interpreted as clusters. Classically the value of the parameters are $k = 2$ and $r = 2$. Then the MCL algorithm can be written as it appears in Fig 3.6.

Input: graph G
Parameter: integer k
Parameter: real r
add loops in G
M_1 = stochastic_matrix(G)
while(diff)
$M_2 = (M_1)^k$ (expansion)
$M_1 = \Gamma_r(M_2)$ (inflation)
diff = difference(M_1, M_2)
end while

Figure 3.6: MCL algorithm

Explained in the language of stochastic flux, the expansion increase the flux inside the clusters while the inflation phase remove the flux between the clusters. Expansion and inflation are to be used alternatively until a steady state is

reached. A steady state corresponds to a double idempotent matrix, *i.e* a matrix that does not change with any more application of both the expansion and the inflation operator. The graph associated with such a matrix is composed of oriented connected components. Each component is interpreted as a cluster, is star-shaped, with an attractor at the center and every member of the component linking to the attractor. Clusters with more than one attractor may appear or nodes belonging to several clusters. The fact that clusters may be overlapping can be an advantage in certain situations where nodes must be connected to several clusters without belonging to any of them.

Regarding the convergence, it can be proved that the process simulated by the MCL algorithm converges in a quadratic time to a steady state. In practice the algorithm stars to converge after a few iterations (around ten). The global convergence is something very hard to prove but it can be conjectured that the algorithm always converges when the input graph is symmetric. As for today there is no mathematical evidence that the algorithm actually converges.

A really important thing about this algorithm is that it identifies the clustered structure of the graph *via* the trace left by this structure on the flux process. The algorithm is fast and support huge graphs (up to hundreds of thousands of nodes, the limitation being the matrix multiplication). Both parameters k and r allow to set the clusters granularity. Mathematics associated with the MCL algorithm show that there is an close relationship between the simulated process and the clustering structure of the graph. The formulation of the algorithm is simple and elegant.

From the definition of this algorithm, one can see that it is really different from others based on links systems. Indeed its behaviour is probabilistic and its termination based on the notion of convergence of a random process. But still this algorithm needs to perform matrix multiplications and therefore cannot be used on the Web graph. I will present lightweight clustering methods in chapter 5 that regroup nodes to fight Webspam.

CHAPTER 4

MALICIOUS BEHAVIOURS IN SOCIAL NETWORKS

This chapter presents SpotRank, which is our answer to malicious users on social news websites. SpotRank is a robust voting scheme for social news websites and the name of the website that implements this scheme.

This chapter first introduces the design of the SpotRank algorithm which is a set of heuristic techniques whose objective is not to detect and suppress malicious voting behaviors in social news website, but rather demote the effects of these behaviors, thus leading to lower the interest of such manipulations for spammers. SpotRank is built over *ad-hoc* statistical filters, a collusion detection mechanism and also over the computation of the *pertinence* of voters and proposed news.

In a second time, I present a strong experimental analysis, a website that implements this algorithm and shows evidence of the efficiency of the approach, both from a statistical and human point of view. This analysis is twofold: I give evidence that using SpotRank maintains a behavior for the social news website which corresponds to a regular behavior, and also provide a study of the perceived quality of the algorithm on 114 users of the experimental platform (*via* a comparison with others french social news websites).

4.1 SpotRank algorithm

In this section I first present the framework on which Spotrank is built together with its principle. I then describe independently each step of the algorithm.

4.1.1 Framework and principle

In this chapter, I consider that the voting system (SpotRank) is used by a community of users belonging to the set $\mathcal{U}$. Any user of the community can propose his own news (or content), which we will call *spots*. The set of spots is denoted by $\mathcal{S}$. Any user of the community can vote for a spot. A vote is a triple (u, s, v) where $u, v \in \mathcal{U}$ and $s \in \mathcal{S}$. The set of all votes is noted $\mathcal{V}$. For the sake of clarity we introduce some notations:

$$\mathcal{V}_u = \{(w, s, v) \in \mathcal{V} \mid w = u\}$$

$$\mathcal{V}_{uu'} = \{(w, s, v) \in \mathcal{V} \mid w = u, v = u'\}$$

$$\mathcal{V}_s = \{(u, t, v) \in \mathcal{V} \mid t = s\}$$

$\mathcal{V}_u$ denotes votes made by u, $\mathcal{V}_{uu'}$ the votes made by u for a spot proposed by u', and $\mathcal{V}_s$ the vote made by all users in favor of the spot s. This modeling is rather theoretical, in practice we have access to a lot more information on users, votes and spots. In the following, we will access this information through functions that will be either clearly defined by the context or explicitly just before their use.

I can now schematize the SpotRank method. It is important to know that it is based on two key notions. The first one is that two votes do not necessarily have the same value. Indeed, each vote will be assigned, depending on many factors, a score. This will induce a score for each spot (the sum of the score of each vote in favor of this spot). The higher the score of a spot, the closer to the first place (e.g., the top of the front page) is the spot. A large part of SpotRank is the score computation mechanism. The other key notion is the *pertinence*. The pertinence of a user depends on the pertinence of the spots he voted for, and *vice versa*. A part of the score's update of a spot will depend on the pertinence of the user that votes for this spot. Last, an additional mechanism is used in order to avoid large scale manipulations: a method to detect cabals (*ie.*, group of users that repeatedly vote one for each other).

Finally, figure 4.1 depicts the voting process of SpotRank for a given spot. The method is working as follows:

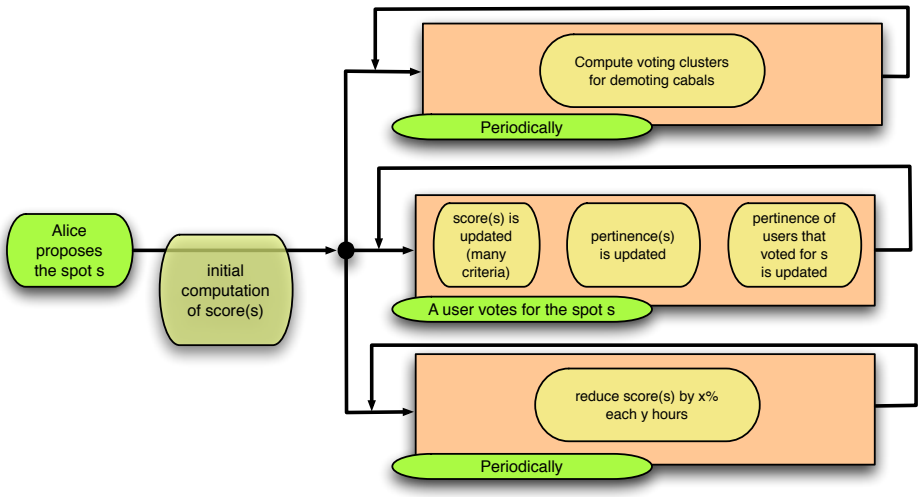


Figure 4.1: Principle of our method

A user proposes a spot. The score of this spot is initialized according to several criteria (all related to the known behavior of the user).
Users vote for the spot. Each vote induces an update on the spot's score and pertinence, but also of the pertinence of users that previously voted for this spot. The score of the spot is then used by a social news website in order to rank published content.
Periodically, an algorithm that detects collusion between users outputs clusters of potential malicious users. These clusters are used for computing the score of a vote.
To avoid old very strong spots to stay forever at the top of the ranking, the score of a spot is reduced **periodically**.

In the following I describe in details each of these steps. Each time a practical parameter is used, it will be denoted by α_i (where $i \in \mathbb{N}$) and its real actual value in practical applications will be discussed. Actual values are set according to a back and forth method: a first value is set, the robustness of the method with this value is tested, the value is modified if necessary, until a good quality of result is reached.

4.1.2 Proposing a spot

The first step of the method is the proposition of a spot. The main threat at this point for social news websites is the system flooding by some malicious users.

When a user proposes a spot it is necessary to initialize its score. In an ideal world any value can be used for this initialization, but this step will be used to demote spots proposed by frenetic spot posters. The initial score will then depend on two factors: first, the frequency at which the user proposes spots and second, the frequency at which new spots come from the user's IP address.

More precisely, the initial score of a spot s is calculated with the following formula:

$$\text{init\_score}(s) = f(n) * c_{IP}(m),$$

where n is the number of spots proposed by the user in the last 24 hours, m is the number of spots previously posted from the user's IP in the last 20 minutes. f and c_{IP} are defined as follows:

$$f(n) = \begin{cases} 100 & \text{if } n < \alpha_0 \\ 50 & \text{if } \alpha_0 \leq n < 2\alpha_0 \\ 10 & \text{if } 2\alpha_0 \leq n < 4\alpha_0 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{and } c_{IP}(m) = \max\left(0, 1 - \frac{m}{\alpha_1}\right).$$

In practical applications α_0 and α_1 will be small integers that depend on the number of visitors of the social news website. I recommend to use $\alpha_0 = 2$ and $\alpha_1 = 10$.

With this formula to initially set the score of a new spot, the effective "spot bombing" from spammers is prevented. Indeed the proposed spot's initial score will drop fast until it has value zero. The $c_{IP}(m)$ coefficient also helps to track down spammers that use several accounts to pollute the spot pool. If we add to a social news website a mandatory identification in order to propose a spot, it becomes meaningless to use several accounts (IP spoofing is then useless).

4.1.3 Voting for a spot

Once a spot has been proposed to the community it can be "pushed" to the front page (*ie.*, put in the top ranked news). The spots ranking is done according to their scores. The voting part is the one requiring the most attention since it is

where the spammers will concentrate all their attacks. I propose a set of filters whose aim is to counter all the attacks a spammer could think of. The idea here is simple: a vote has a base value which is the pertinence of the voter. This value is then modified according to several criteria to provide the actual value of a vote (its score). The score of the vote is then added to the current score of the spot in order to obtain its new score.

Base value of a vote: *pertinence*. The key notion of the voting system is the *pertinence* of users and spots. The pertinence of user u_i is denoted by $pert(u_i)$. Similarly, the pertinence of a spot s_j is $pert(s_j)$.

Definition 2 *The pertinence of a user without voting history (ie., a new user) is a constant* α_3.

The pertinence of a user u_i *with a voting history is:*

$$pert(u_i) = \frac{1}{|\mathcal{V}_u|} \sum_{(u,s,v)\in\mathcal{V}_u} pert(s),$$

where

$$pert(s) = \frac{\text{score}(s)}{|\mathcal{V}_s|}.$$

The pertinence of a user u is thus the average value of the pertinence of the spots u voted for. The pertinence of a spot is its score divided by the number of votes it received (*ie.*, this is the average value of the votes for s). In the experiments $\alpha_3 = 100$ to match the value of a fresh legitimate spot.

It is now possible to define the base value of a vote as the pertinence of the user that votes. This value is weighted by several coefficients that are described in the following, each of these coefficients can be seen as a response to a specific type of attack.

High frequency voting. Most of the time, spammers try to promote a lot of low quality news. All the gain of their manipulations is more due to mass effect than to the promotion of only one peculiar web page. Thus they have no other way of voting than using burst voting. It means that a typical spammer votes for a lot of spots in a short amount of time. To demote this effect the score of a spot is weighted by a coefficient $\text{freq}(u)$ that depends on the user's voting frequency.

Definition 3 *Let* $n = |\mathcal{V}_u|$, $\text{freq}(u)$ *is defined as*

$$\text{freq}(u) = \begin{cases} 1 & \text{if } n < 2 \\ \min(1, \frac{\text{date}(v_n) - \text{date}(v_1)}{\alpha_4 * n}) & \text{otherwise} \end{cases}$$

α_4 is the time interval that is reasonable between two votes.

Abusive one-way voting. In order to decoy manipulations' detection, a typical spammer uses several accounts: one clean account to propose spots, and several disposable accounts to vote for the spots proposed by the clean account. To reduce the score of votes made in this spirit, a coefficient $fp(u, u')$ is defined to take into account the particular frequency of systematic one-way voting from a user u to a user u'.

Definition 4 *Let $u, u' \in \mathcal{U}$, fp is defined as*

$$fp(u, u') = 1 - \frac{|\mathcal{V}_{uu'}|}{|\mathcal{V}_u|}$$

With this coefficient, users that vote only for one specific user will see their vote becoming useless.

Quick voting. Spamming is a large scale activity so it is unthinkable for a spammer to stay a long time on one given website. Their behavior is then to propose a spot and to quickly vote for it using several accounts. This is not a natural behavior since the time interval between the proposition and the vote is too short for a human to even look at the website associated to the spot.

To avoid quick voting any vote is blocked in the first minute of appearance of the spot s on the site and after that the value of the vote depends on a stair function based on date(s) and t the current time. This function (called time) is defined as follows.

Definition 5

$$\mathrm{time}(s) = \begin{cases} 0.3 & \text{if } t - \mathrm{date}(s) < 120 \\ 0.5 & \text{if } t - \mathrm{date}(s) < 240 \\ 0.7 & \text{if } t - \mathrm{date}(s) < 420 \\ 0.9 & \text{if } t - \mathrm{date}(s) < 540 \\ 1 & \text{otherwise} \end{cases}$$

Multiple avatars and physical communities. As said previously, a typical spammer will have many accounts, sometimes he will also have automatic voting mechanisms. These voting bots are often located on only a few servers, so they share the same IP address (or only very few IP addresses). It is then interesting to have a coefficient that demotes votes for a given spot if they come from the same IP address. One could object that legitimate users can share the same IP address. The opinion on this matter is that when legitimate users belonging to

the same IP address vote for the same spot it is a kind of manipulation (one can think of students of the same university that vote for one of them).

Therefore, for a same spot, votes coming from the same IP address receive a decreasing coefficient coeff_{IP} depending on the number n of previous votes from this IP address:

Definition 6

$$\mathrm{coeff}_{IP} = (\alpha_5)^n$$

In practical applications α_5 is a real number between 0 and 1 (typically $\frac{2}{3}$ in this case).

Avoiding the voting list effect. The main threat for social news websites is the existence of cabals. A cabal is a group of people that unite their efforts in order to promote their own spots. There exists highly organised cabals whose goal is to manipulate ranking of social news websites. This is classically done through daily mailing lists. The daily mail contains a list of news for which votes are required. Users of such lists can propose their own news to the list depending, most of the time, of the number of votes they made for other members of the cabal.

In the next subsection I will present the method for detecting cabals (that are called clusters). But here I assume that I already detect these clusters. Such a cluster is a list of users that periodically vote one for another.

To slow down the effect of the cabals I proceed as follows: if a user u votes for a user u' and both users are in the same "cluster" then the value of the vote is weighted by the inverse of the size of this "cluster". This leads to the definition of the following coefficient.

Definition 7 *Let* $\mathrm{cluster}(u)$ *be the cluster to which user* u *belongs (if any), then:*

$$\mathrm{clust}(u, u') = \begin{cases} 1 & \text{if } \mathrm{cluster}(u) \neq \mathrm{cluster}(u') \\ \frac{1}{|\mathrm{cluster}(u)|} & \text{otherwise} \end{cases}$$

Summary: computation of the actual score of a vote. We can now define the score of a vote. This is actually the base value score weighted by all the coefficients defined above.

Definition 8 *The score of a vote* v *from the user* u *for the spot* s *posted by the user* u' *is:*

$$\mathrm{score}(v) = pert(u) \cdot \mathrm{freq}(u) \cdot fp(u, u') \cdot \mathrm{clust}(u, u') \cdot \mathrm{time}(s) \cdot \mathrm{coeff}_{IP}$$

It now remains to define the score of a spot according to this definition.
Computation of the score of a spot. The score of a spot is simply the sum of all votes' score for this spot and of the initial score of the spot. This quantity is however weighted by a multiplicative coefficient that depends on the age of the spot. This time decay is used to promote new spots against old strong spots (it is not interesting that popular news stay forever on top of the ranking).

Definition 9 *The score of the spot $s \in \mathcal{S}$ is defined as:*

$$\text{score}(s) = \text{time\_decay}(s) \cdot \big(\text{init\_score}(s) + \sum_{v \in \mathcal{V}_s} \text{score}(v)\big)$$

The score of a spot s is updated each time a user votes for it, but also periodically since the value of time_decay varies over time. In practical applications, I define the time decay as:

Definition 10 *Let d be the age (in days) of the spot.*

$$\text{time\_decay}(s) = \begin{cases} 1 & \text{if } d \leq 2 \\ 0.8^d & \text{if } d > 2 \end{cases}$$

The decay starts after 2 days to give fresh spots an advantage over old ones. The value 0.8 comes from an experiment on the best value to ensure a sufficient turn-over on the front-page of the social news website.

On a classical social news website using SpotRank, all spots are ranked according to their score (the higher the better).

4.1.4 Detecting cabals

In this subsection, I present the algorithm for the detection of collusion of voters. It is a fair idea to use the weighted directed graph of votes (nodes represent voters, arcs correspond to votes and weights to the number of votes between two users). The standard approach to identify groups of users in this graph is then to cluster it. State-of-the-art techniques such as the ones of [94, 35] are not efficient on large graphs. The graph underlying a social news website can quickly attain a huge size, so I cannot use these clustering techniques.

Instead, I propose here to regroup people that massively vote between themselves. Therefore I use the algorithm presented in Fig. 4.2 that should be run regularly to identify new cabals and actualize the existing ones.

Collusion detection for user u
1. Let $E = \{(v, k)/(\mathcal{V}_{u,v} \neq \emptyset \wedge k = |\mathcal{V}_{u,v}|\}$
   Sort E according to k in decreasing order.
2. Put the first p users (according to this sorting) into a set $Fav_p(u)$.
3. Sort alphabetically the set $Fav_p(u)$ with u's ID included.
4. $\forall v \in Fav_p(u)$, if $|Fav_p(u) \cap Fav_p(v)| > \alpha_6$ then u and v are in the same group.

Figure 4.2: Clustering algorithm

This algorithm should be run for each user in order to assign this particular user to a cluster (potentially a cluster of size 1). Let $n = |\mathcal{U}|$, the complexity of this algorithm is $\mathcal{O}(n \log(n))$ for the first step, $\mathcal{O}(1)$ for the step 2, $\mathcal{O}((p + 1) \log(p + 1))$ for the step 3, and $\mathcal{O}(p^2)$ for the step 4. The total complexity is then $\mathcal{O}(n \log(n) + p^2)$ which is, in practice, $\mathcal{O}(n \log(n))$ since p is a fixed parameter. In the experiment, $p = 5$ and $\alpha_6 = 3$. Those parameters have rather small values, it can be explained by the fact that our community is still small (200 users).

After running this algorithm I store the groups along with their size to reuse this information during the voting phase.

4.2 Experiments

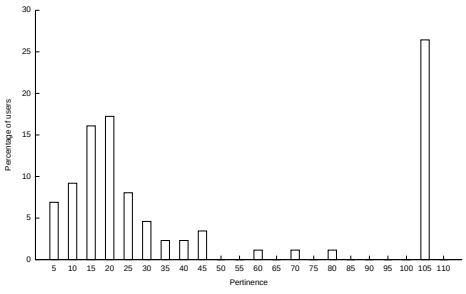
I now present two different evaluations of SpotRank. The first one is a statistical analysis of the output of the method: distribution of users, votes, pertinence of both users and spots, ranking, cluster sizes, etc... The second evaluation is a investigation of the human perceived quality of the ranking method.

In order to collect data about the behavior of SpotRank, Guillaume Peyronnet created a social news website (`http://www.spotrank.fr`). Spotrank.fr strictly implements the method presented in this paper and ranks the spots according to their score (computed as presented in section 4.1). The website has been launched on July the 9th 2009. The data used for this chapter were collected the 10/26/2009. During this period, the website received around 15600 visits, had served around 43000 page views. The bounce rate is 67.85% and the average time spent by a visitor on the website is 2:37 minutes. The presence of spammers

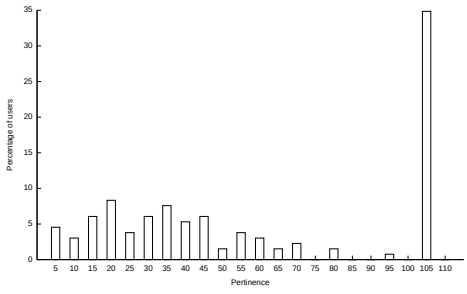
is effective, we estimate (by hand) that at least 10 to 15% of the 200 registered accounts belong to spammers.

4.2.1 Log analysis of spotrank.fr

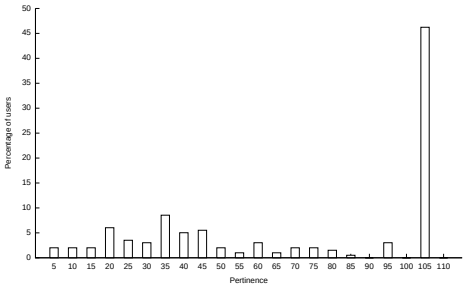
The log files contain all the information about spots and users and allows to show accurately the behavior of the method in a real adversarial environment. The analysis is presented through several figures. The information used for this experimentation was collected between the 23rd of July 2009 and the 26th of October 2009. All the results of the experimentation together with their analysis are presented in the following pages.



(a) 1/3 - 07/23/09



(b) 2/3 - 09/08/09



(c) 3/3 - 10/26/09

Figure 4.3: % of users w.r.t. pertinence

Figures 4.3a, 4.3b and 4.3c. These figures show similar data at three different moments of our analysis. They all represent the percentage of users of SpotRank that have a given pertinence. For instance the first bar on the left of figure 4.3a means that around 7% of the users have a pertinence between 0 and 5. As time goes and the number of users grows the pertinence of the users tends to spread more. The percentage of users at the right of all graphics represents the new users that never voted for anything. Even if the distribution of users regarding the pertinence seems almost uniform, we can see that most users have a pertinence between 15 and 50. Thus we define two categories of users, the non-relevant users whose pertinence is ≤ 10 and the relevant users whose pertinence is ≥ 50 but don't include the newcomers. Being in the non-relevant category does not mean a user votes for spots others don't like. It means that the user votes often for spots with low pertinence. It is likely that this category contains mainly spammers.

Spammers have a low pertinence

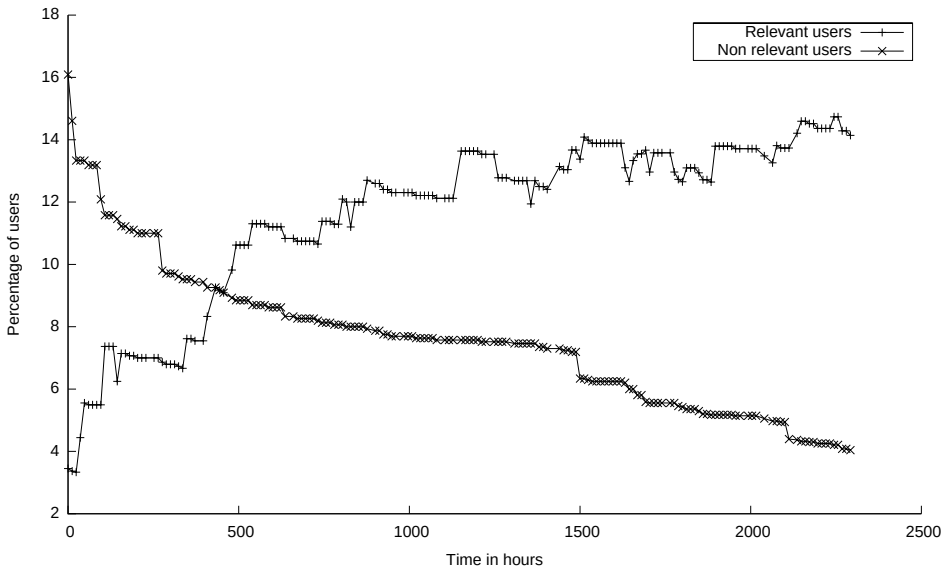


Figure 4.4: % of low and high pertinent users w.r.t. time

Figure 4.4. The two curves on this figure show the evolution of the proportion of users belonging to both the non-relevant and the relevant categories on a period of 3 months. We can see that the percentage of non-relevant users including spammers is decreasing while the percentage of relevant users is increasing. This could be explained by spammers being discouraged but more likely by the arrival of new relevant users. The increase of the size of the relevant category can also be explained by the fact that at the launch of the website, spammers propose spots faster than legitimate users. This means that at first most of the spots have a low pertinence, inducing a low pertinence for voters.

Spammers' weight decrease over time

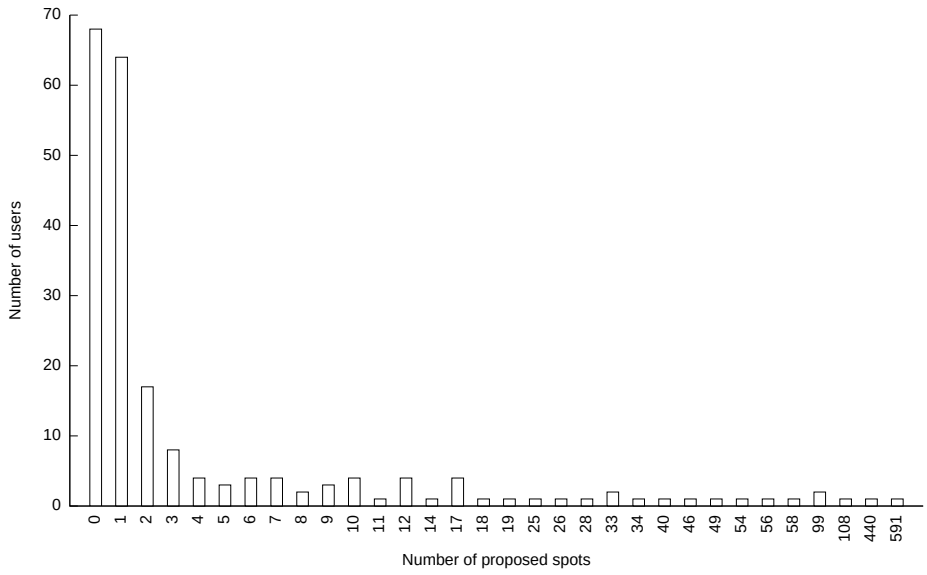


Figure 4.5: # users versus # proposed spots

Figure 4.5. This histogram presents the number of users that proposed a given number of spots during the 3 months of the experimentation. It can be seen that the majority of users proposes a few spots (less than 3). There are few people with an oddly high number of proposed spots. Top proposers were checked by hand. Amongst them, the first three (with respectively 591, 440 and 108 spots) are clearly spammers while the fourth (99 spots) is a borderline user that proposes every single post of his blog.

Spammers propose many spots

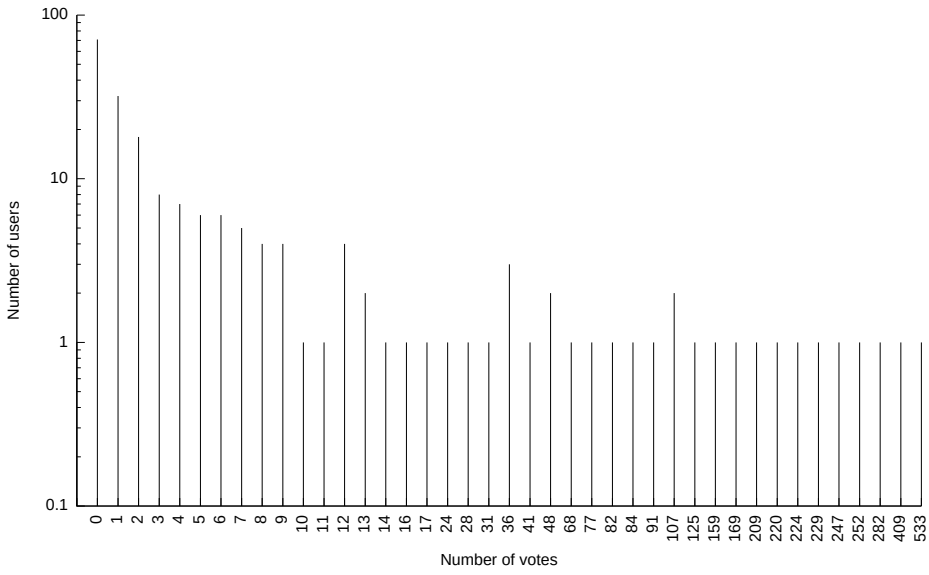


Figure 4.6: % users w.r.t. # votes

Figure 4.6. The figure depicts the number of users w.r.t. the number of votes. Pay attention to the fact that the Y-scale is logarithmic. I can see that most users don't vote a lot: more than 80% of them had voted less than 10 times. The people that vote the most are clearly the ones suspected to be spammers. If we look for instance at the outlier with 533 votes, this is without surprise the spammer spotted on the previous figure. He is a user that proposes a lot of spots and vote only for his own spots. It does not appear in the figures but there are users that do not propose spots (or only very few) and that vote a lot.

Spammers use burst voting

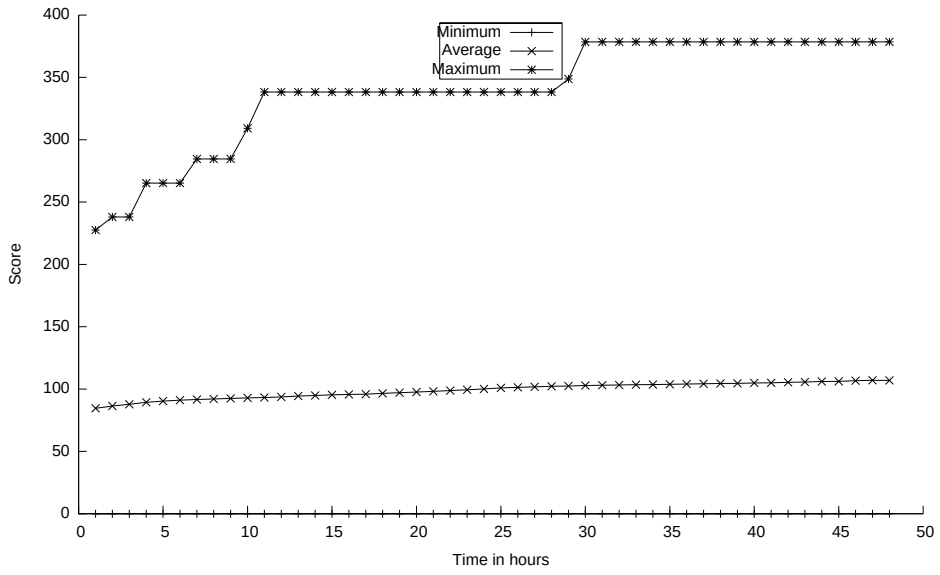


Figure 4.7: spots score w.r.t. time

Figure 4.7. On a social news website such as spotrank.fr there are only a few slots available for promoting highly popular spots. Is this a problem? Is the pressure on spots too high to ensure a fair access to the front page for spots that deserve it? This figure gives the answer by showing the behavior of spots' score during their first 48h of existence (*ie.*, before they undergo the exemption owed to the time decay). The max (resp. min) curve gives the score of the most (resp. least) popular spot at the time the measure is done. The average curve gives the mean value of scores during the first 48h. We can clearly see that only a few spots become popular. It is also possible to infer from this figure the average value of votes for the most popular spot. Indeed, most popular spots received around 15 votes, meaning that the average score of votes for these popular spots is around 25 (to compare to 100, the maximum possible score of a vote, but also to 10, the threshold beyond which users are not considered relevant anymore).

Few spots reach the front page

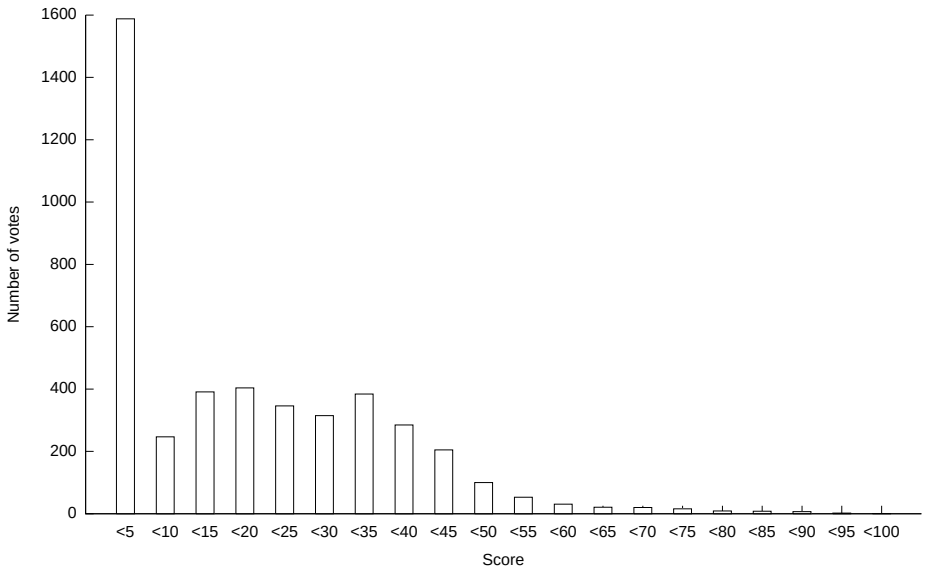


Figure 4.8: # votes versus their scores

Figure 4.8. This last figure shows the number of votes that share a given score. It is clear that most of the votes (≈1600) have very low score. It is not a surprise since we already have exhibited a spammer with 533 votes that vote only for himself and with high frequency, meaning that several of the filters act to preclude the score to be high. Most legitimate users seems to have votes with scores between 5 and 50, and only a few have very high score.

To summarize, the figures provided in this subsection show clear evidence of the effectiveness of the method: spammers are detected, and the score of votes seems to be adapted to avoid manipulations.

Filters reduce spammers' votes' value

4.2.2 Human Evaluation

Even with a very strong analysis of the log files, it is impossible to judge the filtering quality of the method. Indeed, the algorithm consists in filtering news w.r.t. the way people vote, it is not content related. To cope with this issue we decided to gather some feedback from the website visitors. Since an absolute judgement is impossible to obtain without a long debate on what is the quality of a website, we choose to compare the top "stories" of three social news website. The first one is of course spotrank.fr which implements the method presented in this paper, the other two are two of the three french major competitors in the field. The interest of the two chosen social news websites is that they use automatic methods to filter news, and also that the human moderation main goal is to suppress non legal content. The way these two competitors are filtering news is mostly unknown since those are business websites. It is just known that one of them is giving more weight to news for which there are a lot of votes in a short time interval. From now on, they will be denoted as comp1 and comp2. The survey protocol is the following. To have relevant results we periodically collected the first five spots on spotrank.fr together with the top 5 of the two major french social news websites comp1 and comp2. Then disposable Web pages containing a shuffle of this list of 15 news are automatically generated. Each Web page is then sent to a volunteer who has to tell for each news if,

Yes, it is relevant for the news to appear on the front page of a social news website.
No, it is not relevant for the news to appear on the front page of a social news website.
DnK, he is not able to determine if the news deserve to be on the front page or not.
Err, The news was not accessible when he tried.

The first five news of each website were collected during a period of 3 months, and 114 persons participated to the poll. I now present the experimental results.

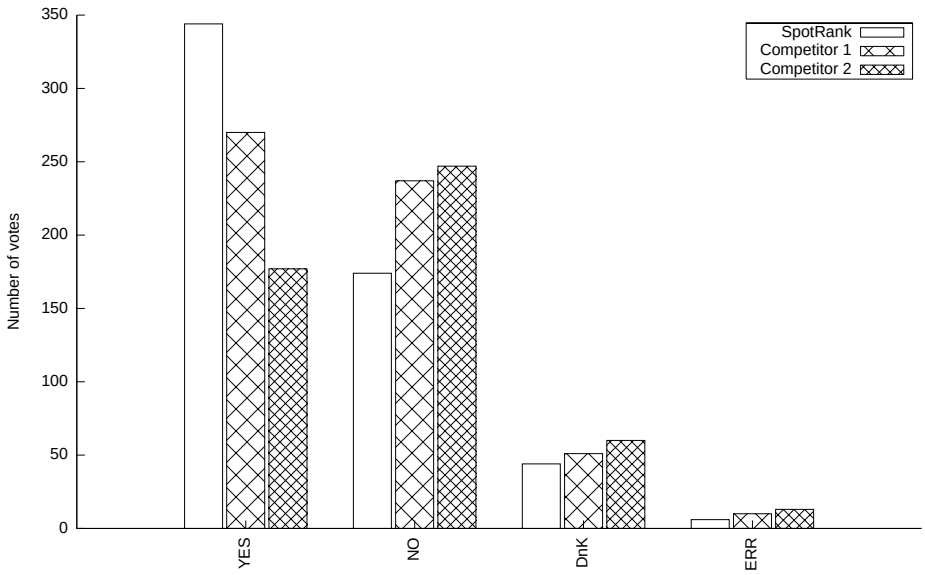


Figure 4.9: # answers of each type

Figure 4.9. This figure could be considered as a summary of the results of the poll. For each competitor it presents the number of **Yes**, **No**, **Dnk** and **ERR**. The number of **ERR** that appears in surveyed people answers is not of interest since this is an external factor that applies for all three social news websites. However a higher rate of error could indicate links to unreliable site. Pay attention to the fact that for each competitor, each surveyed person is giving 15 answers, so the total number of answers is 1710.

SpotRank received 344 **Yes** while comp1 and comp2 received respectively 270 and 177 **Yes**. The performances of comp1 (resp. comp2) are only 78.5% (resp. 50%) of those of SpotRank, thus surveyed people think that the ranking given by SpotRank is of higher quality than the two others. Concerning the **No** answer the situation is similar: this time the lower is the better since this means that the top spots are considered not legitimate and SpotRank received 174 **No**, while comp1 and comp2 received 237 and 247 such answers. Last, 44 **DnK** were received by SpotRank. This is again a better achievement than comp1 and comp2 and this means that the filtering of SpotRank gives clearer results (only a few borderline spots).

SpotRank outperforms its competitors

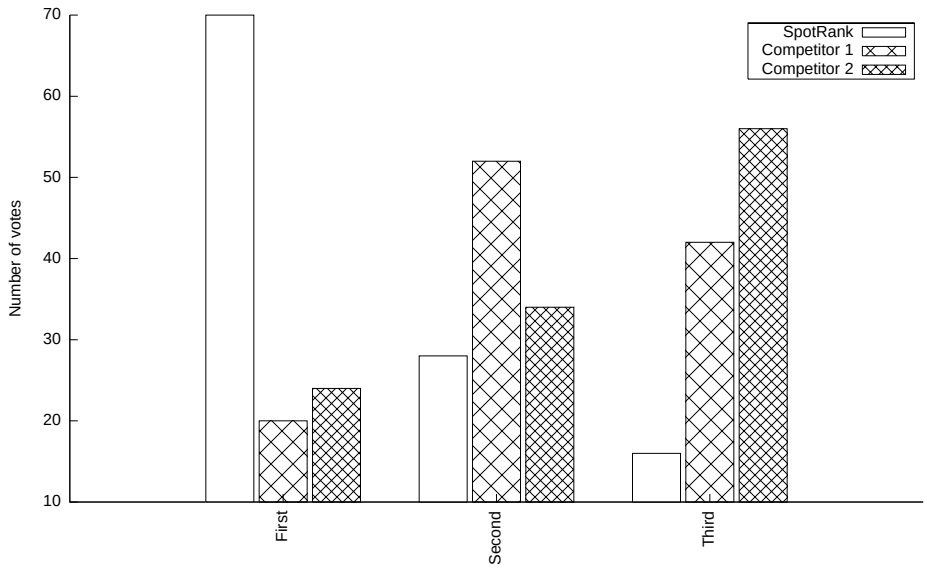
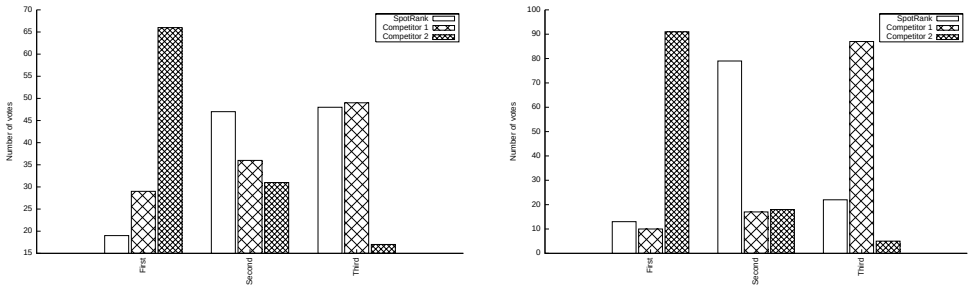


Figure 4.10: rank w.r.t. the number of **Yes**

Figure 4.10. In the previous figure, the behaviors of all users were merged in the counting of each type of answer while here we consider the opinion of each surveyed person. A social news website is considered first if the number of **Yes** he received from a peculiar answerer is greater than those received by the two other competitors. It is second (resp. third) in the case where it received the second (resp. third) number of **Yes**. SpotRank is in first position two times (resp. four times) more than comp1 (resp. comp2), showing again its higher performances. It is naturally second and third less often than the others.

SpotRank's has more legitimate news



(a) rank w.r.t. the number of **No** (b) rank w.r.t. the number of **DnK**

Figures 4.11a and 4.11b. These figures are similar to the one mentioned before for the **No** and **DnK** answers. These two figures confirm what have been presented previously.

To summarize, this user satisfaction survey show clearly that the filtering of SpotRank is perceived to be of high quality. It is interesting to note that some early adopted spammers have already given up playing with SpotRank (see previous subsection).

Spammers can not reach the top 5

4.3 Discussion

This section discuss the interest of the approach made in this paper since statistical filters are an ad hoc response to the spammers problems on social sites. The idea was to prevent identified attacks of spammers on such sites.

Arrow impossibility theorem [3] stated that it is impossible to build a voting scheme with the concurrent fairness properties: unanimity, non-imposition, universality, monotonicity, independence of irrelevant alternatives and non dictatorship (no user can force his ranking to be the global ranking). The last one is exactly what spammers are trying to achieve so it is very important to make their task as hard as possible.

Obviously we will never be able to stop a spammer with infinite resources. In this case he will be able to pay a sufficient amount of different people to produce and vote for his content.

Our method, with the parameters set to their actual value, works fine on a small community as the results presented in this chapter confirm it. The parameters values should be adapted to the size of the community. If one spammer where to find a breach in the set of exposed filters and successfully promotes spam content to the front page, we will have to identify how he proceeds and then create the appropriate filter to void the effects of his manipulation of the system. The method is therefore easily adaptable to any new attack we can encounter.

4.4 Conclusion

In this chapter was presented a robust voting system for social news websites whose goal is to demote the effects of manipulations. Through a website that implements this algorithm, evidence of the efficiency of the approach was shown, both from a statistical and human point of view. SpotRank clearly outperforms real competitors in a real life Web ecosystem, proving the interest of the key notions used to design the method: pertinence, frequency filtering and collusion detection.

CHAPTER 5

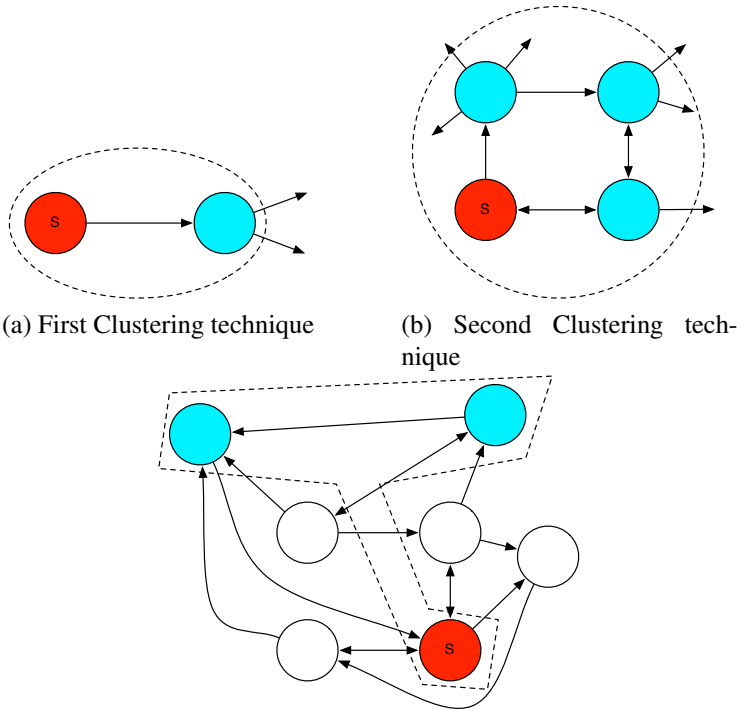
DEMOTING WEBSPAM

This chapter proposes an approach to demote the effects of Webspam through the use of clustering. As explained in chapter 3, graph clustering regroups nodes that are densely connected with few connections between groups. My working hypothesis is that using a clustered version of the Web graph is useful to demote cheaters. This idea to use clustering to fight Webspam is motivated by several facts,

- Spam farms are densely connected regions on the Web and make only few connections to the outside world. Indeed spammers make the PageRank circulates inside their structures to make it grow but are not ready to give it away to a page that does not belong to them. They can be seen as already clustered regions of the Web.
- Pages with a genuinely high pagerank must get incoming links from many places over the Web. There are no reasons for those pages to be densely connected with all their contributors.

Thus if we compute a PageRank where the contributors of node j are nodes i with $i \rightarrow j$ and cluster$(i) \neq$ cluster(j), it should demote the pagerank of cheaters way more than pagerank obtained in a honest manner.

The first part of this chapter introduces the techniques we propose in order to regroup nodes into clusters in a fast and local manner. Then I present the



(a) First Clustering technique

(b) Second Clustering technique

(c) Third Clustering technique

Figure 5.1: Lightweight clustering techniques

experiments that validate my working hypothesis. Finally some statistical evidence is given about the good behavior and the efficiency of one the techniques introduced.

5.1 Clustering Methods

In this section I present the three graph clustering techniques used in our experiments. First, it is important to notice that classical and efficient clustering methods for small graphs such as *the Markov Cluster Algorithm* (MCL, see 3.6.2) and the *edge betweenness clustering* method (EBC, see 3.6.1) are unsuitable for the Web graph because of its size. Indeed MCL requires an explicit matrix representation of the graph which is totally infeasible in our case and EBC runs in time and space of $\mathcal{O}(nm)$ where n is the number of nodes in the graph and m the number of edges (these notations will be kept throughout the chapter), which

is in practice totally non tractable. Moreover we are not interested in an exact clustering. Our interest is not the detection of Webspam but its demotion. If we can group enough Webspam with the target page, building big enough communities, we hope to stop a sufficient amount of incoming pagerank to nullify the Webspam's effects.

Google has indexed more than 1000 billion pages[1]. So in order to be tractable, every technique must have a low complexity, *i.e* linear at maximum. Indeed the PageRank has a linear complexity $\mathcal{O}(n + m)$. Since the PageRank must be calculated to offer a ranking to users, every method which purpose is to demote the effects of Webspam must add at most a constant amount of calculation to be effective. The ideal case would be a method that could be embedded with the PageRank computation with only a constant overhead and no memory usage.

All three methods presented below are local algorithms computed for every node in the graph. The idea is always to have a very simple criterion to group nodes together, starting from a peculiar node. It is also important to use only local knowledge to compute the clusters.

The first technique can be embedded for free during the computation of PageRank. Figure 5.1 shows example of how nodes are regrouped for each method. The red node labelled S is the starting point of the algorithm and the blue nodes are the ones regrouped with it after the computation.

The first method proposed is very intuitive. It simply regroups nodes with one outgoing link with the target of this link (see figure 5.1a). This means that people that give all their pagerank to one person are regrouped with this person. Indeed a well-known technique to raise the pagerank of one page is to create a lot of dummy pages that will give all their pagerank to the target page. This first method will be referred to as Tech1 in the following.

In the second technique, the objective is to regroup nodes that belong to short loops in the graph. In figure 5.1b the length of the loop is 4. For every node in the graph we compute every path of length k and if the path ends on the starting node then everybody in the loop goes into the same cluster. Spammers don't like to waste pagerank, thus many links coming out the target page should return to the target page in a few steps. In the experiments the length of loops was set to $k = 3$. This value was chosen for a tractability motive. It will prove the intuition useful while staying fastly computable. In the rest of this chapter this method will be called Tech2.

The last method simply launches r random walks of length l from every

[1] `http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html`

node in the graph. If the number of random walks that end on a particular node is higher than a threshold t then this node and the starting node are regrouped in the same cluster. With this approach we hope to regroup Webspam with their target page even if some links may lead elsewhere to avoid automatic detection of well known structures. Following links from a Webspam page will lead to the target page with high probability. Later in the chapter this technique will be referred to as Tech3. During our experiments 200 random walks of length 15 were launched. The threshold was fixed at 40 meaning that more than 1/5 of the walks must end on the same node for it to be regrouped with the starting node.

More formally at the beginning of each algorithm every node belongs to its own cluster. When nodes are regrouped we simply merge their clusters following the expression "Any friend of yours is a friend of mine". This has no impact if the starting point of the algorithm has no neighbors in the cluster of the node it wants to regroup with but, on the other hand if it wants to regroup with someone who is very close to one of its successors the starting node probably wants to associate itself with that particular neighbor. Thus two nodes can end up in the same cluster even if the method did not explicitly regroup them.

5.2 Experiments

	Webgraph	*spam*	*nonspam*	*undecided*
		‰	‰	‰
PageRank	84 015 567.786	6.16	50.35	2
Tech1	68 943 484.072	6.13	50.03	2.05
Tech2	48 431 264.361	6.09	47.97	2.01
Tech3	75 176 329.382	6.14	50.67	2

Table 5.1: Pageranks of each set

In this section are presented experiments that have been conducted on the dataset WEBSPAM-UK2007[2]. This dataset is a crawl of the .uk domain made in May 2007. It is composed of 105 896 555 nodes. These nodes belong to 114

[2]Yahoo! Research: "Web Spam Collections".
http://barcelona.research.yahoo.net/webspam/datasets/ Crawled by the Laboratory of Web Algorithmics, University of Milan, http://law.dsi.unimi.it/. URLs retrieved 05 2007.

	Total		Intersection		
	Nombre	Score	Nombre	Score	Percentage
PageRank	56	104 580	46	89 726	100
Tech1	66	85 451.5	46	74 794.4	83.36
PageRank	56	104 580	36	87 877.8	100
Tech2	49	59 648.9	36	42 502.1	48.37
PageRank	56	104 580	52	99 496	100
Tech3	65	92 937.9	52	84 089.2	84.52

(a) Top 20%

	Total		Intersection		
	Nombre	Score	Nombre	Score	Percentage
PageRank	186	155 798	147	128 351	100
Tech1	210	127 193	147	111 833	87.13
PageRank	186	155 798	142	136 765	100
Tech2	203	88 731	142	63 700.1	46.58
PageRank	186	155 798	161	143 803	100
Tech3	189	138 824	161	128 264	89.19

(b) Top 30%

Table 5.2: Effects of differents tests on spam tagged pages

529 hosts and 6 478 of these hosts have been tagged. Please pay attention to the fact that hosts are tagged, not pages (e.g. entire domains instead of peculiar pages). We use the Webgraph [12] version of the dataset by Boldi and Vigna since it allows to manipulate huge graphs without using a lot of memory.

The tagged hostnames are separated into 3 user-evaluated categories: *spam* (690 972 nodes), *nonspam* (5 314 671 nodes) and *undecided* (201 205 nodes). Using this information we can construct 3 sets of Web pages corresponding to the 3 categories.

We first evaluate the pagerank of each node of our dataset. Then we sort each set *spam* , *nonspam* and *undecided* in decreasing pagerank value.

Then we apply each technique to the graph before computing a modified version of the PageRank where i contributes to the pagerank of j iff

- $i \rightarrow j$
- i and j are in separate clusters

	Total		Intersection		
	Nombre	Score	Nombre	Score	Percentage
PageRank	958	846 644	799	699 145	100
Tech1	1 049	690 274	799	596 293	85.29
PageRank	958	846 644	312	389 323	100
Tech2	538	465 120	312	331 698	85.2
PageRank	958	846 644	763	714 624	100
Tech3	830	762 651	763	728 457	101.94

(a) Top 20%

	Total		Intersection		
	Nombre	Score	Nombre	Score	Percentage
PageRank	2 433	1 269 460	1 901	1 062 620	100
Tech1	2 776	1 035 060	1 901	892 642	84
Pagerank	2 433	1 269 460	1 030	745 638	100
Tech2	1 605	697 270	1 030	547 989	73.49
PageRank	2 433	1 269 460	2 028	1 087 260	100
Tech3	2 166	1 142 930	2 028	1 102 770	101.43

(b) Top 30%

Table 5.3: Effects of differents tests on nonspam tagged pages

The contribution C_{ij} of i to j is the following:

$$C_{ij} = \frac{Pr(i)}{k_i} \quad \text{where} \quad k_i = |\{j | i \to j, \text{cl}(i) \neq \text{cl}(j)\}|$$

This is the same as running the PageRank on a graph where all intra clusters edges have been removed. Results can be found in table 5.1. We do not use the normalized version of the PageRank where they all add up to 1 since we make the computation over a huge graph and don't want to be limited by the machine precision. All ‰ in table 5.1 don't add up to one since we consider only a fraction ($\sim$ 5.86%) of all pages (the tagged web pages). It can be seen in this table that every method make the pagerank of all three sets drop. This is easily understandable. Since many edges are removed from the graph, the pagerank cannot spread as easily.

Tech1 reduces the whole pagerank of the graph by $\sim 18\%$, Tech2 reduces it by almost 43% and Tech3 by $\sim 10\%$. We can see that proportions of each set

	Total		Intersection		
	Nombre	Score	Nombre	Score	Percentage
PageRank	49	34 156.5	42	29 705.4	100
Tech1	48	28 748.5	42	26 154.3	88.05
PageRank	49	34 156.5	16	13 851.7	100
Tech2	23	19 875.7	16	14 271.6	103.03
PageRank	49	34 156.5	38	27 011.7	100
Tech3	45	30 774.8	38	26 741.8	99

(a) Top 20%

	Total		Intersection		
	Nombre	Score	Nombre	Score	Percentage
PageRank	109	50 569.9	96	45 923.6	100
Tech1	118	42 566.2	96	39 577.6	86.18
PageRank	109	50 569.9	49	31 952.9	100
Tech2	66	29 427.1	49	25 522.6	79.88
PageRank	109	50 569.9	85	41 865.5	100
Tech3	111	45 230.8	85	41 002	97.94

(b) Top 30%

Table 5.4: Effects of differents tests on undecided tagged pages

is mostly respected using whatever technique. Tech2 is the only one to reduce some set's pagerank. Since it is the *nonspam* set (reduced by more than 2‰) this is not a good news. Tech3 slightly increases the *nonspam* pagerank while slightly decreasing the *spam* pagerank. Obviously this table does not provide enough information to take some conclusions.

Let's take a closer look on the results. We focus on nodes with a proportionately high pagerank (nodes well ranked). We will concentrate the analysis on the first 20% (resp. 30%) of each set, meaning nodes representing 20% (resp. 30%) of the set pagerank. Tables 5.2, 5.3 and 5.4 represent for each technique the number of nodes and score for the whole 20 (resp 30) top percent of each set and the number of nodes and score for the intersection with the 20 (resp 30) top percent of the pagerank of the set. It is important to ensure that the demotion observed at the whole graph level is not uniformly distributed amongst all nodes.

Table 5.2 presents the results for the *spam* set. Regarding the top 20% of this set we can see that for Tech1 and Tech3 there are more nodes in this top 20 than

for the PageRank meaning that each node is weaker. There are fewer nodes for the Tech2.

> Looking at the intersection of each set we can observe that Tech1 demotes the intersection's pagerank by less than 17% which is less than the general demotion registered for this method. Tech2 demotes the pagerank by more than 51% which is better than the general reduction meaning that this spam is actually demoted. Tech3 performs a demotion of almost 15.5% on the intersection which is also better than the general demotion on the whole graph.
>
> On the top 30% of the *spam* set, Tech2 improves its results up to a 53% demotion more than 10 points above the average demotion. Tech1 results worsen to almost 13% and Tech3's efficiency falls to the average demotion.
>
> Now let us focus on the intersections for the 30 top percent. It is interesting for us to have big enough intersection in this case to be sure that strong demoted *spam* nodes are not replaced by stronger promoted *spam* nodes. The size of all intersections combined with the fraction of the set's pagerank they represent allow us to be sure that it is the case.

We can see in this table that only two methods (Tech2 and Tech3) succeed in *spam* demotion. Tech1 has a trend at promoting important *spam* pages.

Table 5.3 shows the results for the *nonspam* set. It is important here to confirm the good results obtained by both Tech2 and Tech3.

> Let's first study the results of Tech1. We observe that it has more pages in both top 20 and top 30 percent of the *nonspam* set. This shows that those pages are weaker and hence demoted. For the top 20% (resp. 30%), Tech1 demotes the intersection by 14.7% (resp. 16%) which represents a small promotion compared to the general demotion. The score on the top 30% is actually worse than the one for the top 30% of *spam* pages.
>
> Tech2 has the smallest number of pages composing the top 20 and 30 percent of the *nonspam* set. This means that the pages are stronger after the application of this method than before. Tech2 demotes the intersection of the top 20% (resp. 30%) by 14.8% (resp. 26.5%) which is way less than the average demotion on the whole graph. This means that these *nonspam* pages are promoted compared to the rest of the graph.

Tech3 also has a smaller number of pages than the PageRank in its top 20 and 30 percent for the *nonspam* set ensuring that those pages have a higher pagerank on average. On this particular set, Tech3 realises negatives demotions *ie* promotions of respectively almost 2% for the top 20% and $\sim 1,43\%$ for the top 30%. These of course are better results than those observed on the whole graph since albeit the general graph lost some pagerank, those particular pages gained some.

Looking at the intersections we can see that Tech1 and Tech3 have large enough intersections but that Tech2 has a smaller one compared to its intersection on the *spam* set. This is of less harm here since we are less concerned about the promotion of *nonspam* nodes but we would like to keep the same sorting as the PageRank as much as possible. The size of this intersection can be explained by the fact that at the top level the fraction of pagerank represented by the *nonspam* set after Tech2 has been applied is smaller than the one of the PageRank, meaning that some important nodes have been demoted since the number of nodes is the same. We are allowed to think then that the ranking of Tech2 may preserve an important part of the PageRank ranking on the *nonspam* set.

Tech3 outperforms Tech2 on this table but it was the contrary on the *spam* table. It is of interest to see how they can be ranked and if the *undecided* set can be helpful to do that.

Table 5.4 concerns the *undecided* set. This set is the smallest and the one that contains the least relevant information since pages contained in this set were not clearly identified as either *spam* or *nonspam* pages.

Our first method continues to produce the same effect previously seen on the first two sets. Meaning the demotion observed on the top 20% (resp. 30%) is $\sim 12\%$ (resp. 13.8%), being inferior to the average demotion. We can then conclude that this technique slightly increase the pagerank of already high ranked nodes and demotes poorly ranked nodes.

Tech2 promotes the top 20% intersection of the *undecided* set by more than 3% but if we consider the top 30% there actually is a demotion of 20% which is less than the average observed on the whole graph.

Tech3 practically does not touch to the pagerank of *undecided* nodes. The registered demotions for the top 20 and 30 percent are respectively of 1% and 2%. These results are again way above the average results of this method.

	Dem.	Prom.	Total
nonspam	458	572	1030
spam	98	44	142
Total	556	616	1172

(a) Observed values

	Dem.	Prom.	Total
nonspam	488.63	541.37	1030
spam	67.37	74.63	142
Total	556	616	1172

(b) Expected values

Table 5.5: Tech2

	Dem.	Prom.	Total
nonspam	86	1942	2028
spam	7	154	161
Total	93	2096	2189

(a) Observed values

	Dem.	Prom.	Total
nonspam	86.16	1941.84	2028
spam	6.84	154.16	161
Total	93	2096	2189

(b) Expected values

Table 5.6: Tech3

The results obtained by Tech2 and Tech3 could be explained by the fact that these sites are borderline. it means that some may be *spam* while other are *nonspam* nodes. Thus, some nodes use the techniques tracked by our methods while others don't. This is clearly visible for Tech2 where we have a promotion on the top 20% but a demotion on the top 30%. Moreover we can see that the number of pages almost triple between the top 20 and 30 percent meaning that there is a gap in pagerank.

Analysing the effects of our three approaches on the *spam* , *nonspam* and *undecided* sets made us realise that Tech1 is not helpful but that Tech2 and Tech3 succeeded in demoting the effects of Webspam while promoting honest pages. Tech2 outperforms Tech3 concerning the demotion of Webspam and *vice versa* regarding *nonspam* promotion. In the next section we will use statistical tools to check whether these techniques are significantly efficient.

5.3 Statistical Test

In this section, we are looking for statistical evidence of the efficiency of our methods to ensure that they are working heuristics. We saw that Tech2 and Tech3 have different effects on pages based on their set of origin. We want to make sure that it is not just a huge coincidence but that it is in fact our methods

	Dem.	Prom.	Total
nonspam	1.92	1.73	3.65
spam	13.93	12.57	26.51
Total	15.85	14.31	**30.16**

(a) Tech2

	Dem.	Prom.	Total
nonspam	0	0	0
spam	0	0	0
Total	0	0	**0**

(b) Tech3

Table 5.7: χ^2 values

that effectively help to separate web pages. We will use a χ^2 independence test to verify that fact.

Here we are only interested in pages with high pagerank before and after the computation of one of our method on the graph. We want to see how these pages are treated by Tech2 and Tech3. We make two categories, pages that are demoted (meaning their particular demotion is greater than the average one) and pages that are promoted (their particular demotion is either negative or less or equal to the average one). Since we are only interested in pages with high pageranks, our sample for each set will be the top 30%.

The hypothesis $\mathcal{H}_0$ we want to test is that both *spam* and *nonspam* pages share the same distribution.

The values V_{ij} for each set and each category can be found in table 5.5a for Tech2 and table 5.6a for Tech3. All categories fill the minimum requirements for the χ^2 test. Table 5.5b and 5.6b show the expected values calculated with the following formula:

$$E_{ij} = \frac{S_{i*} * S_{*j}}{S_{**}}$$

where S_{i*} is the sum of the i^{th} line S_{*j} the sum of the j^{th} column and S_{**} the sum over the lines and columns.

Finally the χ^2 value for both tests can be found in table 5.7a and table 5.7b respectively. Since this χ^2 test is made over 2 categories and 2 sets of values, the critical value to exceed is 3.84 if we want to reject the hypothesis $\mathcal{H}_0$ with a probability of error of 5%. The χ^2 value is calculated according to the following formula:

$$\chi^2 = \sum_{i,j} \chi^2_{ij} \qquad \text{where} \qquad \chi^2_{ij} = \frac{(V_{ij} - E_{ij})^2}{E_{ij}}$$

The χ^2 value obtained for Tech2 is **30.16** meaning that we can reject the hypothesis $\mathcal{H}_0$ with at most 0.5% chances of being wrong. Thus it can be stated that *spam* and *nonspam* pages do not share the same distribution in this case *i.e* Tech2 effectively separates *spam* pages from *nonspam* ones.

The score for Tech3 leaves no doubt, **0** meaning that the two samples share the same distribution. As well as it seems to work in practice, there is no statistical evidence that Tech3 may be able to tell the difference between *spam* pages and *nonspam* ones.

We were only able to show statistical evidence for the good behaviour of one of our technique, leaving us with Tech3 just as an heuristic that seems to work.

5.4 Conclusion

In this chapter we have presented different clustering methods for the demotion of the effects of Webspam on the PageRank algorithm. All three approximate methods are fast to compute and need only a small amount of memory. The last two techniques, based respectively on the identification of small circuits in the graph and random walks, are shown to have good results on Webspam demotion. Moreover, for the method Tech2, we have statistical evidence that it can separate spam and nonspam nodes. The complexity of this method is $\mathcal{O}(n + m)$. Thus this fully automatic method could be effectively added to the already existing arsenal for the Webspam detection and demotion of a search engine. It is still of interest to investigate other methods to perform approximate clustering on the Web graph.

CHAPTER 6

DETECTING WEBSPAM

In the previous chapter, I presented a methodology to automatically demote Webspam using lightweight clustering methods. But it is also of interest to clearly identify people benefiting from Webspam. This is a computationally different problem, the objective is not to compute a fairer ranking of pages but to spot people that do not deserve their rank.

In this chapter I present a method whose goal is to identify malicious structures amongst Web pages. The intuition behind this method is that spammers use specific pages architecture to route the PageRank around the target page in order to maximise its score while avoiding automatic detection. Since the PageRank is related to the behaviour of a random surfer, it seems that using random walks to reproduce the behaviour of this random surfer it will be possible to expose paths created by spammers in order to manipulate and increase their pagerank.

The main results of this chapter are:

- A method based on strong statistical results (borrowed from [31]) that allows to identify malicious patterns in the random walks.
- A methodology that classify random walks in similar categories in order to identify spammers and pages benefiting from their manipulations.
- Strong experimental results showing the efficiency of this approach.

This chapter is organised as follow: section 6.1 introduces the method to identify pages benefiting from Webspam. In section 6.2, I detail the experiments verifying the validity of this approach and show the results before concluding in section 6.3.

6.1 Random walks against Webspam

In this section I describe our method, which is designed to detect Webspammers beneficiaries: target pages. It is based on random walks launched from suspected nodes in the graph. We then analyse how the PageRank is driven through the neighbourhood of those suspected nodes. Since the Web graph is very large, it is of the utmost importance to develop lightweight techniques to detect or demote Webspam. Random walks seems a natural choice for that purpose since they only induce a constant additional cost. Indeed, the crawler of the search engine must already cover the Web graph. The intuition behind the method presented in this chapter is that spammers use specific pages architectures to route the PageRank around the target page in order to maximise its score while avoiding automatic detection.

The PageRank (see the seminal paper of Page *et al.* [80]) simulates the behaviour of a random surfer. This random surfer has two possible choices when visiting a page. He can either follow a link chosen uniformly at random on the page or he can teleport himself to another page on the graph. Spammers can not influence the teleportation but they can drive around the random surfer so he will come back quickly. Since the PageRank of a given Web page is basically the probability of being on that Web page at any moment of a random walk, this procedure will boost the pagerank of this given page. Thus using random walks to reproduce the behaviour of the random surfer we will be able to expose specific paths created by spammers in order to manipulate the random surfer and increase their target page's pagerank.

We want to identify patterns in the random walks. We need now to define the patterns we will look for and use a confirmed methodology to classify random walks in similar categories in order to identify spammers and pages benefiting from spam.

During the random walks we need to store information that can be used at a global level to identify patterns. Using nodes' id won't be sufficient because we won't be able to draw patterns from such specific information. We need less personal information about nodes while exploring the graph. We chose to

associate nodes with their distance from the starting node of the random walk. We limit the distance to a constant d meaning that all nodes which distance dist_i to the starting node i is such that $\mathrm{dist}_i > d$ then $\mathrm{dist}_i = d + 1$. This neighbourhood is called the d-neighbourhood of the node i. Thus if we consider the neighbourhood of distance at most d the language has $d+2$ symbols $[0 \cdots d+1]$. This step is illustrated in Fig. 6.1a. Using distances instead of nodes' id will make it possible to draw pattern and regroup nodes with similar random walk. It will also help to recognize structures independently from their size. If structures serve the same objective they will have similar walks even if the sizes differ.

The information we are interested in during the random walks is how the PageRank is driven through the different levels of the neighbourhood. Thus we focus on the n-grams in the random walks. We then use the so-called ustat_k vectors on the random walks.

For a word w over the alphabet A it is possible to compute the vector $\mathrm{ustat}_k(w)$ which is a vector of size $|A|^k$ and defined as,

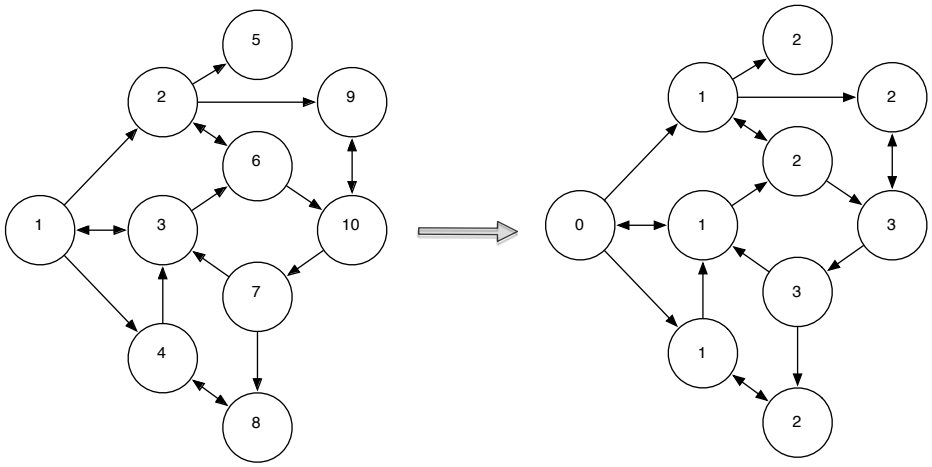
$$\forall p \in [0 \ldots |A|^k - 1], \mathrm{ustat}_k(w)[p] = \frac{\mathrm{occ}_p(w)}{|w| - k + 1}$$

where $\mathrm{occ}_p(w)$ represents the number of appearances of the k-gram p in w and $|w| - k + 1$ the number of blocks of size k in the word w. $\mathrm{ustat}_k(w)[p]$ is thus the frequency of p as a k-gram of w (more details about the theory behind the ustat_k vectors can be found in [31]).

Using all these tools, we will now be able to identify similar structures, *i.e.* structures that produce mostly similar words. If two architectures produce similar words, it means that the PageRank is driven the same way around the target page (source of the random walk). If we can identify how a spammer route the PageRank in his neighbourhood we will be able to compare its ustat_k with the ones computed on suspicious nodes. The key point that makes the use of ustat_k vectors highly effective for this goal is that they are robust, as the following result from [31] states. This proposition deals with the relation between L_1 distance, distance over words and ustat_k vectors.

Proposition 1 *For large enough words $w, w' \in \Sigma^*, \forall \delta > 0$, for large enough k:*

- *if dist$(w, w') \leq \delta^2$, then $||ustat_k(w) - ustat_k(w')||_1 \leq 7 \cdot \delta$.*
- *if $||ustat_k(w) - ustat_k(w')||_1 \leq \delta$ then dist$(w, w') \leq 7 \cdot \delta$.*



(a) Step 1

0	1	2	3	3	2	1	1	0	1	2	3	3	1	0	1	2

(b) Step 2

$$\mathrm{ustat}_2^t = \left(\begin{array}{cccccccccccccccc} 0 & \frac{3}{16} & 0 & 0 & \frac{1}{8} & \frac{1}{16} & \frac{3}{16} & 0 & 0 & \frac{1}{16} & 0 & \frac{1}{8} & 0 & \frac{1}{16} & \frac{1}{16} & \frac{1}{8} \end{array} \right)$$

(c) Step 3

Figure 6.1: Graphic description of our method

Our Algorithm: **Input** G the graph.
Input i the starting node.
Param d the neighbourhood distance.
Param l the length of the random walk.
Param k the size of the k-grams considered.

1. Compute the neighbourhood of distance d.
2. Launch a random walk of length l.
3. Compute the Ustat$_k$ vector associated with the random walk.
4. Compare the vector with the library.

Figure 6.2: Algorithm

Thus if two ustat$_k$ vectors over w and w' are close (in the L_1 sense) then w and w' are very similar.

Fig. 6.2 presents the algorithm derived from the previous proposition. This algorithm is used to match structures crawled amongst the Web graph against a library of previously known spam structures. It is correct thanks to the proposition 1. Fig 6.1 shows the execution of the algorithm on a very small graph shown on the left of Fig. 6.1a. First in Fig. 6.1a, every node is labelled with its respective distance to the node 1 starting point of the random walk. Then we launch a random walk of size 16 resumed in Fig. 6.1b. The statistical projection of this random walk can be seen in Fig. 6.1c. The comparison with the pattern library will occur on the ustat$_k$ vector.

Let's now look at the complexity of the algorithm. The first step is the computation of the neighbourhood of distance d for a node i and costs C_i which is defined as

$$C_i = 1 + \sum_{0 \leq k < d} \sum_{i \rightarrow_k j} d_i^+$$

where $i \rightarrow_k j$ means there exist a shortest path of length k between i and j and d_i^+ is the outdegree of node i. The worst case complexity scenario happens when a node i is able to reach all the nodes within its neighbourhood of distance d inducing a worst case complexity of $C_i = \mathcal{O}(n + m)$.

Summing this complexity for all nodes gives a running time

$$C = \sum_{i \in V} C_i = \mathcal{O}(n^2 + nm)$$

The complexity of launching a random walk of length l is constant for a given node since it consists of $l - 1$ random choices. Random walks should be launched for all suspected nodes. Without previous knowledge on suspected nodes, it means it should be launched on all nodes with high pagerank or at most for every node in the graph. Thus this step requires $\mathcal{O}(n)$ steps.

The computation of the ustat_k vector for a random walk of length l requires also l steps since it requires to scan the whole random walk. Then this step takes $\mathcal{O}(n)$ operations to compute the vectors for all nodes.

The complexity of this algorithm is the complexity of its first step meaning that it can run in time $\mathcal{O}(n^2 + nm)$. This implies that it is impossible to run the algorithm on the whole graph and that one should first select the node he wants to inspect, nodes in a certain range of pagerank for example.

6.2 Experiment

In this section, I present the experiments that have been conducted on the dataset WEBSPAM-UK2007[1]. This dataset is a crawl of the .uk domain made in May 2007. It is composed of 105 896 555 nodes. These nodes belong to 114 529 hosts and 6 478 of these hosts have been tagged. Please pay attention to the fact that hosts are tagged, not pages (e.g. entire domains instead of peculiar pages). We use the Webgraph [12] version of the dataset by Boldi and Vigna since it allows to manipulate huge graphs without using a lot of memory.

The hostnames are tagged with three different labels **spam**, **nonspam** and **undecided**. We are not interested in the last one since it does not provide discriminant enough information. In addition to those two sets we add a third set which we called **spam linked** since it is composed of nodes that are not in the spam set, but whose distance to nodes in the spam set is at most 2. Originally the spam linked set had 309 508 nodes but we remove from this set all the nodes that are also in the nonspam set. The intersection of those two sets contains 11 814 nodes. This is quite important and someone must keep in mind that they are

[1] Yahoo! Research: "Web Spam Collections".
http://barcelona.research.yahoo.net/webspam/datasets/ Crawled by the Laboratory of Web Algorithmics, University of Milan, http://law.dsi.unimi.it/. URLs retrieved 05 2007.

	Nodes		PageRank	
	Value	Percentage	Value	Percentage
Graph	105 896 555	100	84 015 567	100
Spam	690 972	0.65	517 546	0.62
Spam Linked	297 694	**0.28**	7 733 663	**9.21**
Nonspam	5 314 671	5.02	4 230 292	5.04

Table 6.1: Number of nodes and pagerank part for all three sets

probably many other suspicious nodes in the nonspam set. The final size of the spam linked set is then 297 694 nodes.

We then run a PageRank computation on the whole graph. We used the non-normalized version of the algorithm because of computer precision issues and run 60 iterations. The results are shown in table 6.1. We can see in this table that, despite its few nodes, the spam linked set has and oddly high pagerank. This is surely explained by how this set was created. Many nodes have a huge part of their incoming links coming from the spam set. Thus it is surely in this set that we can find pages benefiting from Webspam. The latter probably more represented in the spam set. Indeed, this is a standard behavior for spammers to link from highly "spammy" Web pages their specific targeted (but clean) page.

The next step of these experiments was to launch random walks from every node belonging to one of the three tagged sets. More precisely we launched one random walk for every node and every set of parameters. Indeed we tried neighbourhood of distances 2 and 3 and looked at ustat_k vectors for bi- and trigrams. The presented results are those obtained with a 3-neighbourhood and computing ustat_2 vectors. The size of the vectors is then 25.

First we want to look over each set for general statistics. More precisely, we are interested in knowing the proportion of **sinks** (nodes without outlinks) in each set as well as the number of random walks that lead to pagerank **evasion** *i.e.* the number of random walks that go to the $d + 1$ (here 4) level more often than they come back. The results are shown in both tables 6.2a and 6.2b. We can see that the set that minimizes both *criteria* is the spam linked set. Indeed sinks are clearly obstacles to a good circulation for the PageRank and evasions are not recommended if you want to maximise your pagerank. Note that those evasions may not all be real evasions since There are no differences between levels after 3.

	Number	%
Spam	116 401	16.85
Spam Linked	16 497	5.54
Nonspam	609 307	11.46

(a) Sinks in each set.

	Number	%
Spam	117 579	17.02
Spam Linked	47 654	16. 01
Nonspam	1 005 904	18.93

(b) Evasions in each set.

	Number	%
Spam	8 406	1.22
Spam Linked	88 069	**29.58**
Nonspam	132 931	2.50

(c) Returns to the origin.

Table 6.2: General statistics.

In addition we look at random walks that come back to their origin at some point. People want to maximize their pagerank so they articulate the architecture around their target page to lure the random surfer. It is clear from results in table 6.2c that most cheaters are in the spam linked set. Pages in the spam set are pagerank accumulators and aggregators but the spammers objective is not to maximize the pagerank of those pages, since as said before the target page of webspammers is often a genuine page.

The presented technique proceeds by comparing the frequency vectors computed to already known ones. Thus, we need a pattern library to compare the vectors to. Since the objective is to identify cheating structures, it is important to search for these patterns in the good set. We chose to extract the patterns we will look for in the spam linked set since it is supposed to contains people benefiting from spam.

As said previously, spammers needs to make the pagerank circulates but must ensure that this pagerank will come back really often. We then chose to extract all ustat_2 vectors, from the 100 nodes with the highest pagerank in the spam linked set, that present a high enough frequency of the pattern 01 meaning that the random walk came back to its starting point. We selected random walks that came back at least 5 times to their starting point.

This leads to fourteen patterns we will attempt to identify in all three sets. We consider that a vector v matches a pattern p whenever $|v - p|_1 \leq 0.2$. Note that the maximal distance between two vectors is 2.

The results of these matches can be seen in table 6.3. We can see in this table that they are very few matches for the spam set. This again provides evidence

	Number of matches	PageRank		
		Sum	Highest	Lowest
Spam	43	97.6	13.87	0.47
Spam Linked	**2460**	**107 372.51**	8704.22	0.48
Nonspam	1069	7002.65	517.55	0.36

Table 6.3: Results of comparisons

that nodes in this particular set are not here to benefit from Webspam, but are indeed Webspam and aim at producing artificial PageRank for nodes in the spam linked set.

Regarding the nonspam set. It has quite a few matches. It is reasonable to think that those matches are in fact cheaters since the intersection between the nonspam and the spam linked set is not null. We can see that around 12% of those matches are in fact in the intersection.

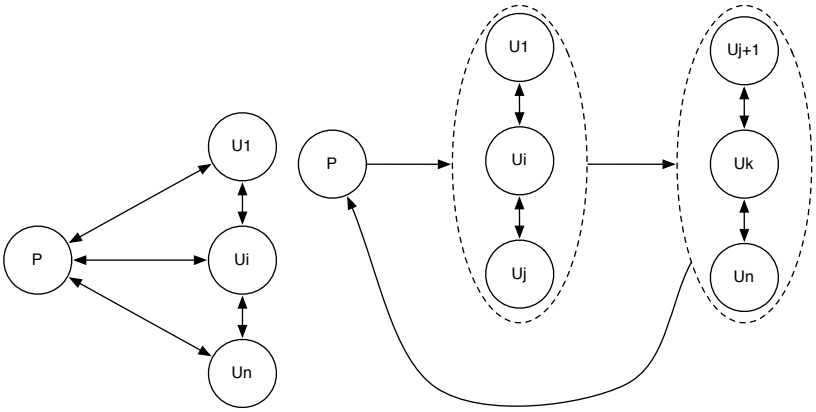
It is in the spam linked set that we find the most matches that are really efficient. With only 2.3 times more matches than the nonspam set, the sum of the pagerank of the matched nodes is more than 15 times bigger.

It is interesting to note that pages whose random walks have a high number of returns to the origin mostly have a high pagerank. Since we can not massively find this pattern in the nonspam set, it can not be considered as a normal behaviour. This means that owners of these pages uses some kind of specific architectures in order to make the PageRank circulates around the page while maximizing the PageRank of the page itself.

Looking at the patterns extracted, it is possible to divide them in 2 categories presented in Fig 6.3. Of course those patterns are subjected to slight changes, loosing a bit of effectiveness but becoming really harder to identify.

In the first pattern (Fig 6.3a), we observe that the PageRank circulates a lot between the neighbors of the target page that exchanges a lot with its neighbors. This pattern may seem natural for a site but since it is rarely identified in the nonspam set, it is clear that only Webspammers set up this particular architecture to maximize their score.

The second pattern shown in Fig 6.3b creates short loops to bring the PageRank back to the target page. The PageRank is driven 2 steps away from the target page before coming back directly. This structure is rather usual in search engine optimization (SEO) since it is widely known by spammers that reciprocal linking is detected (and penalized) by the search engines, and so triangular linking



(a) First pattern (b) Second pattern

Figure 6.3: Patterns extracted from spam linked top100

is then set up to avoid detection.

Based on those promising results, it would be interesting to investigate further the effective patterns we can find in those ustat$_k$ vector. It is possible to find people benefiting directly from Webspam but spammers are getting smarter and their techniques more complicated every day. It could be also of interest to consider "middle men" in the Webspamming world to separate real cheaters from their pagerank providers. Those "middle men" are nodes taking part in smaller structures that boost their pagerank but not too much, then those pages can link directly to the real target page that does not have to make reciprocal links and can avoid detection since it does not appear in a special architecture. The detection of those "middle men" is a lesser evil for spammers since it does not expose the real target page.

To understand in more depth the mechanisms built by spammers, ustat$_k$ vectors for $k > 2$ should be analysed with more precision. Since they give more details about the spammers' strategy for PageRank redirection. They would also give a better mechanism for pattern comparison in terms of precision.

6.3 Conclusion

In this chapter, I presented a technique whose goal is to detect web pages that benefit from Webspam. This technique is built up strong theoretical foundations through the use of the ustat$_k$ vectors of [31]. Experimental results show that this

technique is both effective and efficient since we are able to detect cheaters using a few simple patterns. Moreover, this method is robust towards slight changes in the spam farms since it looks for small distances between ustat_k vectors. This means that with a well constructed pattern library it is possible to catch a lot of cheaters.

However, this technique may not be applied on the whole graph since the neighbourhood computation step may cost too much on graphs with high outdegree nodes. One should select first suspected nodes on several criteria: PageRank range, biased neighbors distribution, etc. Once the set of starting points is selected the method can be applied to detect efficiently the cheaters amongst the suspected pages.

Part II

Analysis & probabilistic solutions for large scale systems

CHAPTER 7

INTRODUCTION

In this part of the thesis, I focus on large-scale networks. The networks I am interested in are well-structured networks whose sizes tend to increase. Sensor networks or distributed applications usually work fine when the number of entities is limited (*i.e.* small). The problem here is to design algorithms that scale-up to hundreds of thousands of entities and beyond. It is also of interest to possess tools to analyse in depth such systems.

Sensor networks are huge networks of small entities used to monitor wild zones. Sensors are nowadays cheap enough to allow the constant growth of networks. The main problem concerning those networks is to have a good scheme of messages propagation to minimize the power consumption. In chapter 8 I describe a new data dissemination protocol for this purpose. This work led to an international publication in the 13^{th} ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM 2010) [17].

I also focus on cluster and grid applications. It is really important for these applications to be fault-tolerant. There are only two ways to ensure this on such applications. Either it is mathematically proved or the design of the application is verified using formal methods. On the other hand you can attest that your application is fault-tolerant through an extensive series of tests or simulations.

A proof (defined by formal methods) or a verification through model checking is doable on really small systems. It is almost impossible to correctly model

an application that will run on many computers (due to the so-called state space explosion phenomenon). Thus the only viable solution is to conduct experiments. Those experiments should be conducted in a totally controlled environment leading to the creation of dedicated platforms to test large-scale applications. I present in chapter 9 a testbed for large-scale applications. This work was presented in the 6th ACM conference on Computing Frontiers (CF 09) [47] and was later extended with the same co-authors in a book chapter [46].

This chapter is structured as follows. In section 7.1 I introduce sensor networks together with our problem and existing solutions. Section 7.2 presents clusters and grids and their challenges. It also presents the difficulty of developing applications for such systems and the huge need for a thorough analysis.

7.1 Message propagation in sensor networks

7.1.1 Sensor Networks

A sensor is a small unit, whose objective is to aggregate data in its environment, transform it into a numeric message and send this message to a sink node if necessary.

Sensors communicates between themselves using wireless techniques. Their neighborhood consist of all nodes inside a certain radius as illustrated in Fig 7.1. The technology used for communication highly influences the capability of the signal to go over obstacles or the power required to send a message.

Since they are small, sensors have high limitations in terms of power and memory. They are autonomous units and embed most of time an non-rechargeable battery, thus power consumption is a real issue when using sensors.

Sensors only make two things, gather data and exchange messages. Exchanging messages consumes much more energy than obtaining data. It is then really important to minimize the number of messages that circulates between the sensors.

The topology of a sensor network is not known by the sensors, they only have a local knowledge of this topology. Those networks are supposed to cover a wide area and sensors are not displayed by hand but more likely uniformly disseminated.

This implies that sensors can land anywhere, so it is impossible to predict the topology of the network. Moreover sensors may fail, others may be added to the

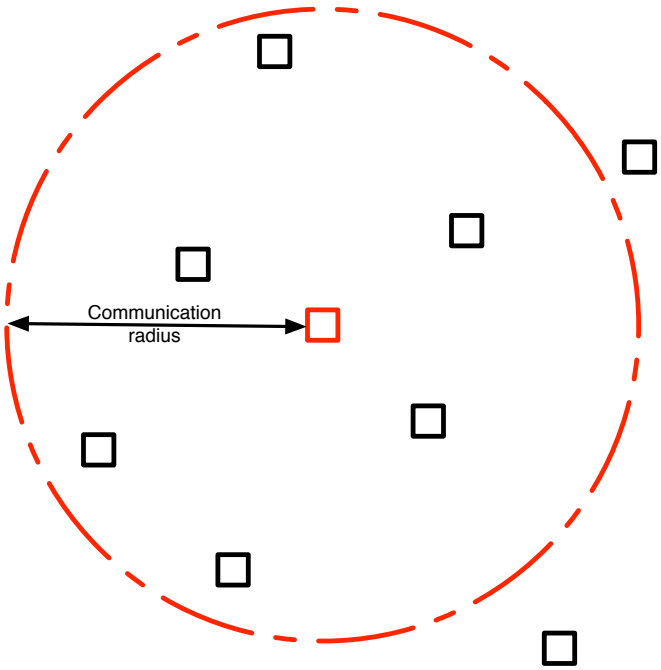


Figure 7.1: Illustration of the communication radius in a wireless sensor network

network implying a topology changing over time. In some cases, more complex sensors may also be able to move, generating a changing topology as well.

Promoted by the military research since they are the ideal tool to monitor a battlefield or verify the sanity of a region before deploying troops, sensor networks were a pretty active field [1, 41, 25]. The idea that during the nineties it was made possible to produce small sensors at a low price is widely spread in the literature. This is really controversial and has not been proved. But it has triggered a huge interest for scalability related algorithmic issues.

Many academic disciplines are interested by sensor networks. The main fields interested by sensor networks are the following, environment, military, health, home and commercial for applications as diverse as fire detection, battlefield surveillance, monitoring a patient's condition without intrusive procedures, buildings weaknesses detection, better management of home automation to save energy, real time tracking of parcel post, ...

7.1.2 How to efficiently propagate messages in sensor networks

The key issue of sensor networks is message propagation. I present in this thesis joint work done to propose a new scheme that reduces the number of messages exchanged to disseminate data in the network. This work can be used by zone monitoring wireless sensor network (WSN) applications: a large number of sensors is deployed to collect data or events in a specified geographic area.

Data collected by sensors is later gathered by an entity called sink node[1], which will store and process the whole network data. A sink can be static or mobile. In the former case, connectivity to at least one sink in the network has to be assured in order to guarantee a good information retrieval. In the later case, the mobile sink has the flexibility to move over the network and gather the collected data, and the multi-hop sink connectivity is not always required [21, 68]. For these reasons, data management reveals to be an important design issue in monitoring-based WSNs.

A key challenge in this context is *how to efficiently distribute and store monitored data such that it can be later sent to or retrieved by a sink*? A failure in this process might result in data loss. Much work has been carried out on data dissemination in WSNs. Basically, these works can be categorized as reactive or proactive approaches. In the reactive approaches, sensors have to react to indications of the position of static sinks or of trajectory taken by mobile sinks, so that their monitored data can be sent accordingly to the sinks location [54, 69, 6]. In proactive approaches, the monitored data should be disseminated through the deployment region in advance so that (i) connected paths may be later established between storing nodes and the static sinks or (ii) the mobile sink may later visit the storing nodes [91, 42, 96]. In the latter case, the way the data dissemination is performed will determine if the sink trajectory may be either predetermined with controlled mobility [97, 42] or free by following an uncontrolled mobility pattern [96]. Fig 7.2 represents a WSN where red nodes between the two curves represents either nodes connected to the sink in the static case or nodes on the sink trajectory in the dynamic case.

Additionally, given the resource-limited characteristics of sensors, any data dissemination mechanism developed for them must be simple and incur low overhead. Assuring *an efficient data dissemination by only using local inferred neighborhood knowledge while respecting resource constraints* is thus an inter-

[1] A sink is a node with no resource limitation.

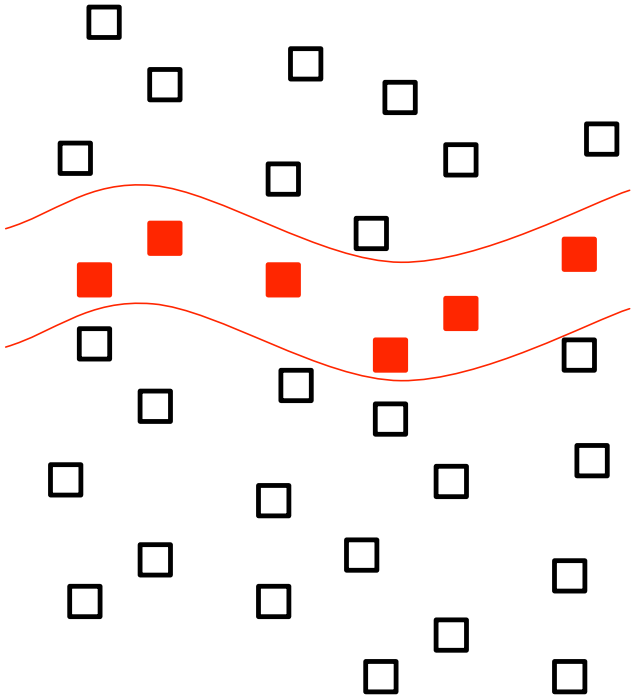


Figure 7.2: WSN with storing nodes represented in red

esting challenge in WSNs and the focus of our research. In fact, the difficulty in selecting well distributed storing nodes in a network strongly depends on the amount of network knowledge available to sensors. If every sensor has a complete knowledge of the network, selecting storing nodes becomes trivial. On the other hand, if sensors are only aware of their neighborhood and have no location information, ensuring that a set of well selected storing nodes in the network emerges from individual decisions is challenging.

Although a large amount of effort has been invested in designing data dissemination algorithms for WSNs [96, 42, 6], the provision of a lightweight data distribution strategy adaptable to any criterion of storing nodes' selection (e.g., storage capability, energy constraints, network location, equal storing load distribution, etc) has not received similar attention.

To counter these issues, join work realised with Aline Carneiro-Viana, Thomas Hérault, Sylvain Peyronnet and Fatiha Zaïdi in chapter 8 presents a flexible proactive data dissemination protocol, called *Supple*.

7.1.3 Related work: data dissemination in WSNs

In the literature, much work has been carried out on data dissemination in WSNs. The way the data dissemination is performed depends, however, on how the sink gathers the monitored data made available by the sensors in the network. In a general point of view, data aggregation allows a structural organization of the network topology. This organization can be: *reactive* or *proactive*.

Surveying the literature, it appears that early research on reactive approaches in wide-area static sensor networks can be traced back to Directed Diffusion [54] and SPIN [44]. [103, 45] are other examples of works dealing with static sink and reactive dissemination strategies. [103] establishes a data collection tree by query propagation from sinks. [45] is based on a isobar mapping, which allows to build a topographic map of a space populated by sensors and data aggregation with nodes similarity depending on collected data. In this last years, [67] has proposed an approach that combines the push and pull queries strategies, known as an hybrid approach. Among the proactive approaches, Ratnasamy et al [91] proposed the use of a Distributed-Hash-Table structure on top of the geographic routing protocol (GPSR) to support data-centric storage. In [43], a clustering-based protocol is proposed to transmit data to the base station.

On the other hand, the presence of sinks that can move and directly collect data from sensor nodes in a monitored area, avoids the necessity for sensor-to-sink path maintenance in the network. [69, 6] are examples of reactive dissemination approaches where the mobile sink follows a controlled, and thus predictable, trajectory. In this case, the sink must visit some predefined nodes to retrieve a representative view of the monitored data. On the other hand, reactive dissemination approaches where the mobile sink follows a free, i.e. uncontrolled, trajectory, requires the sensors to track the sink mobility in order to adapt or influence the data dissemination [101, 56]. In summary, these reactive proposals must dedicate a significant amount of resources to track the sink and to forward on-the-fly the data to be collected towards the mobile sink.

In proactive data dissemination strategies with predictable sink mobility, data is sent by sensors to a well selected subset of nodes, typically forming a virtual structure, to be later retrieved by the mobile sink [97, 42]. On the other hand, if the mobile sink performs an uncontrolled mobility, no structure can be defined. In this case, the dissemination should be performed in a way that allows the sink following a free trajectory to retrieve a representative view of the monitored area by visiting a relatively small number of any nodes in the network [96].

7.2 Large scale applications

7.2.1 Cluster & Grids

The computation needs are always increasing because people are trying to resolve problems with growing sizes. One computer is now not enough to resolve a problem in a reasonable time or can not store the instance of the problem in its memory.

There are two approaches to overcome computation power issue. The first one is to build "supercomputers", huge machines that gather many CPUs and a huge memory. But these computers are very expensive and hard to build.

People then decided to regroup computers in order to sum their capabilities to solve problems. They are several ways to regroup computers to overcome computation and memory issues.

Clusters. The first one is to create clusters, *i.e* groups of identical computers connected through high speed, reliable network. The size of these clusters is nowadays around hundreds of computers. Computers in clusters are all on the same site to reduce time needed to exchange information and increase the reliability of communications.

The idea is to divide the computation between all the computers using several paradigms, distributed or parallel computing for example. To the end user, the cluster is seen as one machine where a computation is sent. Then specific software is use to divide the computation between the nodes in the cluster and gather the result once every body is done. Clusters may also be used for computation and data replication offering some quality of services in case one of the cluster nodes is subject to a failure.

Grids. As stated by Ian Foster in [33], They are three requirements for a system to be considered a grid. These are the requirements as written by Ian Foster,

"

1 *Coordinates resources that are not subject to centralized control ... (A Grid integrates and coordinates resources and users that live within different control domains—for example, the user's desktop vs. central computing; different administrative units of the same company; or different companies; and addresses the issues of security, policy, payment, member-*

ship, and so forth that arise in these settings. Otherwise, we are dealing with a local management system.)

2) *... using standard, open, general-purpose protocols and interfaces ... (A Grid is built from multi-purpose protocols and interfaces that address such fundamental issues as authentication, authorization, resource discovery, and resource access. As I discuss further below, it is important that these protocols and interfaces be standard and open. Otherwise, we are dealing with an application- specific system.)*

3) *... to deliver nontrivial qualities of service. (A Grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating for example to response time, throughput, availability, and security, and/or co-allocation of multiple resource types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts.)*

”

Computation grids are composed of many heterogeneous computers in several geographic sites connected through long-distance unreliable network. The idea behind grids is to utilise every cycle of a computer in the grid maximizing the computation power.

Grids often propose much more raw resources than clusters. The two architectures are not competitors since they are not used for the same purpose. But developing for both architectures implies the same kind of problem.

7.2.2 Managing failures in large-scale systems

One challenge of the utmost importance in large-scale applications is the fault-tolerance. Going from one to several machines to solve one problem creates many issues. Someone has to decide how to divide the computation to create his application.

The resulting application is more prone to failures. Using only one computer, two kinds of fault may appear. The computer may crash thus ending the computation or the result may be incorrect due to memory corruption for example. By coupling machines, communication is introduced. Messages can be lost and/or corrupted which is another source of error. Plus errors can be propagated leading to bigger errors.

When developing a cluster or grid application, someone should take extra-care making his application fault-tolerant. It is really important to guarantee that even in the presence of failures the computation will end correctly in a reasonable amount of time. Obviously when you are using hundreds of computers for several hours exchanging thousands of messages, something will go wrong, computers will crash, messages will be lost and/or corrupted.

It is then an important challenge, extensively covered in the literature [89, 102]. To ensure fault-tolerance, people use various techniques that often consists in an upper layer in communications protocol in order to detect failures or data corruptions.

During such a development, someone must ensure that his application is fault-tolerant. It is really important to get behavioral guarantees on the application. Monitoring an application behavior at large scale has its own problems too. The first one being the resources. Indeed, it is often hard to obtain all the machines needed to realise an experiment, it is harder to obtain them just to test your application since you may need them on a long period of time to run all your tests. Plus you have to control your environment in order to understand what went wrong or why everything went right. The ability to reproduce the conditions is important to ensure that an application passes the exact same tests it has failed before.

7.2.3 Evaluation of large-scale applications

One of the most important issue for the evaluation of a large-scale application is to monitor and control the experimental conditions under which this evaluation is done. This is particularly important when it comes to reproducibility and analysis of observed behavior. In grid software systems, the experimental conditions are diverse and numerous, and can have a significant impact on the performance and behavior of these systems. As a consequence, it is often very difficult to predict from theoretical models what performance will be observed for an application running on a large, heterogeneous and distributed system. It is thus often necessary and insightfull to complement the theoretical evaluation of parallel algorithms with simulations and experiments in the “real world”.

However, even with detailed monitoring procedures, experiments in the real world are often subject to the influence of external events, which can prevent more detailed analysis. More importantly, the experimenters usually have access to only a small variety of distributed systems. In general, experimental conditions are not strictly reproducible in the real world. The approach usually

taken to broaden the scope of the evaluation consists in designing simulators, under which the experimental conditions can be as diverse as necessary. The "real world" experiments can then help to validate the results given by simulators under the reproduced similar conditions.

Still, simulators can only handle a model of the application, and it is hard to validate an implementation, or guarantee that the end user application will meet the predicted performance and behavior. Here, we study another tool for experimentations: emulators. Emulators are a special kind of simulator, which are able to run the final application, under emulated conditions. They do not make the same kind of abstraction as normal simulators, since they emulate the hardware parts of all the components of the real world infrastructure, and thus capture the complex interactions of software and hardware. Yet, since the hardware is emulated in software, the experimenter has some control on the characteristics of the hardware used to run the application.

Through this control, the experimenter can design an ad-hoc system, suitable for his experiments. Of course, the predicted performances must still be validated by comparison with experiments on real world systems, when such systems exist. But within an emulated environment, the experimenter can inject experimental conditions that are not accessible in a real environment, or not controlled. A typical example of such condition is the apparition of hardware failures during the experiment. With a real system, hardware failures are hard to inject, and hard to reproduce. In an emulated environment, hardware is software-controlled and the experimenter can design a reproducible scenario of fault injection to stress fault tolerant applications. This is crucial in fault-tolerant systems, since the impact of the timing and target of a failure can impact tremendously on the liveness and performance of the application.

Emulators can be designed at different levels of the software stack. A promising approach for emulators is the use of virtual machines (VM, see [87]). A VM by itself fits partially the goals of parallel application emulators, since it emulates (potentially multiple) instances of a virtual hardware on a single machine. In addition to these virtual machines, we need to link them through a controllable network. In chapter 9, I present V-DS, a platform for the emulation of parallel and distributed systems (V-DS stands for Virtual Distributed System) through virtualization of the machines and the network.

This platform was initially realised by Benjamin Quétier, Mathieu Jan, Thomas Hérault and Franck Cappello. I added the low-level network emulation with the protocol EtherIP [52] to enable the emulation of grids on clusters.

7.2.4 Existing solutions for large-scale applications development

Recently, the number of large-scale distributed infrastructures has grown. However, these infrastructures usually fall either into the category of production infrastructures, such as EGEE [2] or DEISA [74], or in the category of research infrastructures, such as PlanetLab [22]. To my knowledge, only one of the testbeds in the latter category, namely Grid'5000 [16], meets the mandatory requirement for performing reproducible and accurate experiments: full control of the environment.

For instance, PlanetLab [22] is a good example of testbed lacking the means to control experimental conditions. Nodes are connected over the Internet, and a low software reconfiguration is possible. Therefore, PlanetLab depends on a specific set of real-life conditions, and it is difficult to mimic different hardware infrastructure components and topologies. Consequently, it may be difficult to apply results obtained on PlanetLab to other environments, as pointed out by [40]. Grid'5000 [16] consists of 9 sites geographically distributed throught France. It is an example of a testbed which allows experiments in a controlled and precisely-set environment. It provides tools to reconfigure the full software stack between the hardware and the user on all processors, and reservation capabilities to ensure controllable network conditions during the experiments.

However, much work remains to be carried out for injecting or saving, in an accurate and automatic manner, experimental conditions in order to reproduce experiments.

Finally, Emulab [99] is an emulation platform that offers large-scale virtualization and low-level network emulation. It integrates simulated, emulated and live networks into a common framework, configured and controlled in a consistent manner for repeated research. However, this project focuses only on the full reconfiguration of the network stack. Moreover, Emulab uses extended FreeBSD jails as virtual machines. Inside jails, the operating system is shared between the real machine and the virtual machine, thus killing a virtual machine is the same as killing a process. It means that this framework may not simulate real (physical) machine crashes. To the contrary, our work uses Xen virtual machines, allowing to either shutdown the machine or crash it, which will leave the connections open. As will be demonstrated in the experiments section, this is a much more realistic crash simulation since a crashed machine never closes its

[2] EGEE Team. LCG. `http://lcg.web.cern.ch/`, 2004.

connections before disappearing from the network.

Software environments for enabling large-scale evaluations most closely related to ours are [14] and [77]. [14] is an example of integrated environment for performing large-scale experiments, via emulation, of P2P protocols inside a cluster. The proposed environment allows the experimenter to deploy and run $1,000,000$ peers, but at the price of changes in the source codes of tested prototypes and supporting only Java-based applications. Besides, this work concentrates on evaluating the overhead of the framework itself and not on demonstrating the strength of it by, for instance, evaluating P2P protocols at a large scales. In addition, the project provides a basic and specific API suited for P2P systems only. P2PLab [77] is another environment for performing P2P experiments at large-scale in a (network) controlled environment, through the use of Dummynet [86]. However, as for the previously mentioned project [14], it relies on the operating system scheduler to run several peers per physical node, leading to CPU time unfairness. Modelnet [93] is also based on Dummynet. It uses the same scheme except that the network control nodes do not need to co-scheduled on the compute nodes. In Modelnet, network nodes are called *core nodes* and compute nodes *edge nodes*. But, as in P2PLab, multiple instances of applications are launched simultaneously inside *edge nodes*, consequently it relies again on the operating system to manage several peers.

Finally, simulators, like Simgrid [18], GridSim [15], GangSim [27], OptorSim [7], etc. are often used to study distributed systems. Simgrid [18], which is designed to test scheduling algorithms, makes event-driven simulation where resources are defined through two characteristics, the *latency* and the *service rate*. GridSim [15] is a toolkit based on Java virtual machines and allows to define many parameters during the simulation like the machines locations of define resources time- or space- sharing. OptorSim [7] is also written in Java and its objective is to study the effectiveness of replica optimisation algorithms in grids. Finally GangSim [27] is a simulator that study scheduling strategies on grids with a special attention on resources allocation policies. The main problem with simulation is that it successfully isolates protocols but does this at the expense of accuracy. Some problems that have been overlooked by the abstractions done in the simulation will not be exhibited by simulations but will be observed when the real application is launched, so conclusions from simulation may not be valid, like in [32].

CHAPTER 8

QoS IN SENSOR NETWORKS

This chapter proposes a new data dissemination protocol for sensor networks called Supple. Amongst all scenarios presented in the previous chapter it was chosen to focus on the proactive data dissemination strategy and on how to select well distributed storing nodes in WSN. In this approach, sinks may be then either static and located on the border of the network, or mobile with its trajectory unknown to the sensors.

Supple allows data dissemination to a subset of nodes in the network by following any previously defined selection criterion. One example of selection criterion concerns a subset of border nodes, which can be an interesting storing option if sinks are located close to the network border or follow a controlled trajectory defined by the border nodes location. Otherwise, by uniformly distributing data over the target set $|S| = n$, Supple can be also adapted to the case where the mobile sink performs uncontrolled mobility. Nevertheless, contrarily to the approaches presented in [96, 4], Supple allows an efficient data dissemination with a lower communication overhead.

The principle of the data dissemination scheme Supple is the cladding of a topology over the sensor network to create a specific routing between nodes with a maximal distance in $\mathcal{O}(\log(n))$ steps.

The novelty of Supple is its flexibility in selecting good storing nodes respecting the established selection criterion without having a global network knowl-

edge. Additionally, in a network with n nodes, Supple guarantees that the contact and data gathering by a sink from only m storing nodes, where $m << n$, will allow it to get a representative amount of data of the whole network.

Supple empowers sensors with the ability to make storing decisions that rely on neighborhood information only and flexible selection criteria, which can follow any predetermined distribution law.

8.1 Supple description

Supple is based on three phases: neighborhood discovery, weight distribution, and data dissemination. Sensor nodes use a simple tree-based structure, constructed during the neighborhood discovery phase, which allows weight distribution amongst nodes. Weights are based on a predefined criterion of selection as well as a distribution law, and are used by sensors at the data dissemination phase. At this phase, sensors then make on the fly forwarding and data storing decisions based on their own weights and the weights of their neighbors. Supple takes thus advantage of the bias amongst different sensors' weights for good data dissemination. This behavior can be used for *uniform data distribution*, by assigning equal weights to all sensors as well as for *specific data distribution*, by assigning high weights to specific selected set of sensors. This later distribution can be useful, for instance, in cases where, to avoid network disconnections, only border nodes are used for storing activities: referred here as *location-based data distribution*.

This chapter provides a detailed formal analysis of the implementation of Supple. It then analytically compares Supple with other data dissemination techniques, such as RaWMS [4] and flooding. Although Supple achieves the same properties as RaWMS for uniform data dissemination, it will also be shown it has a much better message complexity (*ie.*, Supple uses exponentially fewer messages than RaWMS). Moreover, Supple has the advantage of being flexible, meaning it allows disseminating data on any subset of the nodes with any distribution. Finally, by simulations, we study the performance of Supple on a large set of topologies and compare it to RaWMS [4]. The simulation results largely confirm the theoretical analysis. In addition, they show Supple is practical and effective in distributing data amongst selected storing nodes that respect the predefined distribution criterion with limited network knowledge. Simulation results also show that Supple outperforms RaWMS in term of robustness to message losses.

8.2 Rationale and system model

The storing selection flexibility feature of Supple allows its combination with data gathering strategies based both on static or mobile sinks, with the condition of accordingly setting the predefined selection criterion. For instance, in the case of static sinks located close to the border of the network, a location-based data dissemination may be used, where only border nodes would perform storing activities.

Case of study. Let's consider an application where a large number n of sensors are randomly scattered on a given geographic area for collecting data or monitoring events. Data can be collected by a finite set of static or mobile sinks.

Nodes. All sensors are uniquely identified. Sensors are all *equal* in terms of computational, memory, and communication capabilities. No synchronization is required. Following the proactive data dissemination's procedures, each sensor node in the network is provided with a *partial view* regarding some other nodes (including itself). In this way, each node may act as a storage node for some other nodes in the WSN, but not for all of them. By slight abuse of terminology, I will use the term *view* both for the actual information stored at a given node p and for the IDs of the nodes whose information is stored at p. The size of views will be analyzed in the following sections. To counter the limited buffer of sensors, we consider the use of power-aware compression algorithms to deal with the main drawback of partial views of s entries at storage nodes [71]. Generally, if the latency is not an application issue the data collected by sensors can be locally compressed before being disseminated, reducing the network traffic and thus prolonging the network lifetime. For instance, algorithms like the one presented in [71] reaches compression rations up to 70% on environmental datasets.

Communication. Each node i is able to wirelessly communicate with a subset of *neighbors* that are within i's transmission range: a transmission disk centered on i with radius t. We assume *bidirectional* communication and that the average density of nodes $d_{avg} = \frac{\pi t^2 n}{a^2}$ is such that the resulting communication network is connected [36]. The network is thus modeled as a 2-dimensional *Unit Disk* graph $G^2 = (n, t)$, being $G = (V, E)$ where V is the set of network nodes and E models the one-to-one neighboring links.

Limited initial knowledge. Initially a node $i \in V$ only knows its identity, the fact that no two nodes have the same identity, and a parameter $W(i)$ that define its weight in the network ($W : V \rightarrow \mathbb{N}$ is called the weighting function). Weights are initially assigned to nodes based on an external criterion of storing nodes'

selection. For uniform selection, all sensors will have the same weight and then, the same chances to be selected as storing node by another node. On the other hand, if the criterion is a location-based selection (e.g. the network border) only nodes at the specified location will be used as storing nodes: each sensor i will have $W(i) = 1$ if it is, for example, located on the border of the network, and $W(i) = 0$ if it is an insider node (cf. [84] for border detection strategies). Supple may also rely on the use of dynamic weights amongst selected storing nodes: e.g. to give to nodes located on the network border and having higher storing or energy capabilities, a higher probability of storing. In this case, an external mechanism should provide this information and accordingly assign the parameter $W(i) > 0$ to each node i. This however, is not the focus of this chapter. Hereafter, I use the term *target set* to refer to the subset of nodes with weight greater than 0, i.e. the storing nodes. Finally, nodes do not know their position and we do not use any geographic knowledge in this algorithm. The presented hereafter approach relies solely on node connectivity.

8.3 Supple: formal presentation

In this section, I formally present the supple algorithm. Supple's goal is to allow each node sending its collected data to a *target set*. Using Supple, it is possible to ensure that each storing node of the target set has a view of controlled size s containing data collected by any s nodes in the network, which are chosen according to the Supple algorithm. The general principle of Supple is described in Fig. 8.1. Further details are provided in the following sections.

8.3.1 Tree construction

Let $G = (V, E)$ be the graph that represents the network. The first step of Supple relies on a tree construction: a tree-based routing structure $T(G)$ initiated by a central-localized node in the network and that is at least binary. The constructed tree $T(G)$ embeds the connectivity of the network and ensures that sampling a node according to a given distribution can be done with a logarithmic number of hops. In particular, Supple requires a bootstrap phase where $T(G)$ is constructed using a cost metric propagated in 1-hop `Hello` messages. The constructed $T(G)$ structure is thus, an aggregation of the shortest paths from each sensor to the central-localized node based on a cost metric, which can represent any application requirement: hop count, loss, delay amongst others. An important set of

Supple Input: Graph $G = (V, E)$
Input: Target set $S \subseteq V$
Input: Weighting function $W : S \rightarrow \mathbb{N}$
Input: View size $s \in [1, \ldots, |V|]$

1. Construction of the tree $T(G)$ from G
2. Propagation of the weights of the target set nodes
3. For each node $i \in V$

   send $data(i)$ to the root of $T(G)$
4. The root propagates each $data(i)$, $r(s)$ times according to the probabilities induced by the weights over the target set

Figure 8.1: Principle of Supple.

routing protocols in WSNs are based on the construction of a tree-based routing topology rooted at the sink [54, 70]. Other tree-based structures that can be used in conjunction with Supple are PeerNet [29] and Tribe [95]. PeerNet [29] in particular, constructs a binary tree. Finally, *Supple can be adaptable to any kind of structure, the only requirement being the routing capability.*

In the rest of this chapter, I consider that the $T(G)$ construction is performed with a hop count metric and that the tree is binary. The complexity of the tree construction is then $\mathcal{O}(n)$. Note that this is done for the sake of clarity and it is not a limitation of our method.

8.3.2 Weight distribution

The flexibility of Supple is given by the fact that the data dissemination can be adapted to any target set (denoted by S). For this, all nodes of the target set have a weight assigned through a function $W : S \rightarrow \mathbb{N}$. The probability of sending data to a particular node i is given by the weight assigned to node i with respect to the sum of all weights of storing nodes in the target set. Although allowing the use of any selection criterion of storing nodes (e.g., uniform, location-based, energy-based, etc), *I consider in this chapter, the uniform selection criterion in order to allow the comparison with the related approach RaWMS [4].* For this

Weight distribution
Input: Tree $T(G)$
Input: Target set $S \subseteq V$
Input: Weighting function $W : S \to \mathbb{N}$

For each node $i \in T(G)$ create a triple $(l_i, W(i), r_i)$
For each node $i \in T(G)$ in a breadth-first search starting from the leaves

do let $j :=$ left child of i in $l_i = l_j + W(j) + r_j$

do let $p :=$ right child of i in $r_i = l_p + W(p) + r_p$

Figure 8.2: Weight distribution in $T(G)$.

reason, $|S| = V$ and for all nodes $i \in S$, $W(i) = 1$. This will give to all nodes the same chances to be selected as storing nodes. Once equal weights are assigned to nodes, they perform the weight distribution over the tree, as depicted in Fig. 8.2. The idea behind this algorithm is to initialize each node $i \in S$ with a triple $(l_i, W(i), r_i)$, where l_i (resp. third component r_i) is the weight of the left (resp. right) subtree of i and $W(i)$ is the weight of the node i in the target set.

It is clear that the complexity of the whole weight distribution process is $\Theta(n)$. Additionally, the weight distribution only requires a field of at most $\log n + \log |W|$ bits in the usual `Hello` packet.

8.3.3 Data dissemination

The data dissemination is the most important phase of Supple. This phase ensures the properly data propagation at storing nodes. Since I consider here that nodes have the same weight, this phase has to ensure an uniform distribution of nodes' data amongst the target set.

The idea is the following. Firstly, all nodes must send their data to the root of the tree (i.e. the node that started the tree construction), as detailed in Fig. 8.4. When the root receives a new data from one of its children, it means that a node is propagating its information for dissemination into the target set. The root propagates then, $r(s)$ times the data to its children. This will ensure the views are of size s (cf. Proposition 2) . The propagation by the root is done according to the weights of its left and right subtrees and also to its own weight (in the

Table 8.1: Comparison between Supple, RaWMS, and flooding.

	# rounds	**msgs per round**	**mem. overhead**	**add. overhead**
Supple	$r(s)$	$n \cdot \log n$	view size s	3 ints per node
RaWMS	$r(s)$	n^2	vew size s	
flooding	1	n^2 broadcasts	linear	flooding mem.

Forward_data
Input: Tree $T(G)$ with a triple $(l_i, W(i), r_i)$ for each $i \in T(G)$
Input: viewsize s
Input: data d

(code for node i)

Pick at random uniformly $x \in [0, l_i + W(i) + r_i]$
If $x < l_i$ then send d to left child
If $l_i \leq x \leq l_i + W(i)$ then store d in own view
If $W(i) + l_i < x$ then send d to right child

Figure 8.3: Algorithm for forwarding data down into the tree.

case the root is also in the target set). The *Forward_data* algorithm depicted in Fig. 8.3 formally presents this local propagation. Moreover, it must be noted that messages are forwarded asynchronously, *i.e.* there is no reason for the root to finish the $r(s)$ sequential data sending of a node to start sending data of another node. Thus, each node forwards messages coming from its parent according to the *Forward_data* algorithm (cf. Fig. 8.3). It is worth noting that the algorithm naturally stops when the message is received by a node whose left and right component of the triple equals to 0.

At the end of the data dissemination, all nodes of the target set will have, with high probability, a view of size s. This view is randomly composed by nodes' data distributed according to the weights given on the target set. In the case weights are equal for all nodes, it naturally achieves view of size s with uniformly disseminated data.

The complexity of the data dissemination is the keypoint of Supple. Each node sends its data to the root, which implies $\mathcal{O}(n \log n)$ messages since nodes

Data dissemination
Input: Tree $T(G)$ with a triple $(l_i, W(i), r_i)$ for each $i \in T(G)$

For each node $i \in V$ do

Send data to its parent

On reception from a child by node i do

if $i \neq$root then forward to parent

if $i =$root then do $r(s)$ times

Forward_data

On reception from the parent by node i do

Forward_data

Figure 8.4: Algorithm for the data dissemination.

are at distance $\mathcal{O}(\log(n))$ from the root at most. Then each data is propagated $r(s)$ times through the tree (from the root to the leaves), resulting in $\mathcal{O}(n \cdot r(s) \cdot \log n)$ messages. Finally, the complexity in term of messages of the whole process is $\mathcal{O}(n \cdot r(s) \cdot \log n)$. In this way, Supple outperforms the message complexity of the related work closest to Supple, the RaWMS[4], which achieves a complexity in term of messages of $\Theta(n^2 \cdot r(s))$.

8.3.3.1 Formal analysis

Here, we address the problem of computing the number of times a message must be sent through the tree in order to ensure that the size of the views will be, with high probability, s. Intuitively, if each disseminated node data would have reached a different storing node, then in order to obtain a view of size s, it would have been enough to start s data sending at each node, during the data dissemination phase. Nevertheless, two data sending started at the same node i have a non-negligible probability of reaching the same storing node j. Thus, in

order to obtain the target view size s, each node should start a larger number $r(s)$ of sending, where $r(s) > s$. The following proposition gives an explicit lower bound for the value of $r(s)$, depending on s.

Proposition 2 (Computation of $r(s)$) *Let n be the number of nodes in the tree $T(G)$. To obtain, with high probability, a view of size s, $r(s)$ messages must be sent, where:*

$$r(s) = \begin{cases} n \ln(\frac{n}{n-s}) & if\ s \neq n, \\ n \ln n & if\ s = n \end{cases}$$

Proof 1 *Let's first consider the case where $s \neq n$. We want to compute the number of messages that must be sent in order to obtain with high probability, s different nodes, performing the data dissemination according to uniformly distributed weights. The number of unreachable nodes p after r(s) messages is: $\mathbb{E}(p) = n - s$, which can be rewritten as*

$$n(1 - \frac{1}{n})^{r(s)} = n - s$$

Using the classic inequality $(1-x)^y \leq e^{-y.x}$ this means:

$$e^{(-r(s)/n)} \geq \frac{n-s}{n}$$

By choosing $r(s) = n \ln(\frac{n}{n-s})$, we obtain the aimed expectation, i.e. balanced views of size s.

Let's now prove the case where $s = n$. In this case we have $n \cdot e^{(-r(s)/n)} = 0$. Using $r(s) = (k+1) \cdot n \ln n$ (with $k \in \mathbb{N}$), we obtain:

$$n \cdot e^{(-r(s)/n)} = n^{-k}.$$

This means that the probability of a collision can be as small as wanted.

Another point of interest is the relationship between the size s of the view and the size $|S|$ of the target set. Indeed, if the target set size $|S|$ is too small with respect to the view size s, not enough space will be available to store data of nodes of the whole network. Hence, even if the data stored by all nodes in the target set is gathered, it will not provide a representative amount of network data. The following proposition addresses the relationship between s and $|S|$.

Proposition 3 (Relationship between s and $|S|$) *Let $|V| = n$. If the view size of each node is limited to s, then the target set S must contain at least $\Theta(\frac{n}{s} \ln n)$ nodes in order to guarantee with high probability a good data storing and a satisfying data gathering by a sink.*

Proof 2 *It uses the solution of the coupon collector's problem that can be found in [73] (pages 57–63). In this case the number of trials x is $s \times |S|$, in [73] it is proven that $x \geq n \ln n$, thus implying $|S| \geq \frac{n}{s} \ln n$.*

The proposition 3 only gives hints to Supple users in order to make sure that the size of the target set is large enough to store the nodes' data of the whole network. In the case where $|S| = n$, the Proposition 2 also gives the *minimum number of storing nodes m in the target set to be contacted by a sink in order to gather a representative amount of data of the whole network.* Note that only the quantity of $(\frac{n}{s} \ln n)$ target nodes has to be respected and the sink is free to contact *any* node in the target set.

8.3.4 Discussion

Here, I discuss the pros and cons of Supple and compare its complexity to the RAWMS strategy. The data dissemination of Supple has the flexibility and the self-organizing feature of being adaptable to any kind of data distribution, this is dictated by the way weights are distributed amongst nodes. In the particular case of uniform distribution, this data dissemination is done more efficiently than in the related approaches, thanks to the tree structure. Indeed, this implies a huge improvement of the number of messages used to obtain the uniform distribution.

Additionally, Supple requires a small additional overhead in term of memory: only a triple of integers. Finally, Supple is robust to messages losses and failures of storing nodes (as stated in section 8.4), since the data of each node is replicated in $r(s)$ storing nodes. If the sink is mobile, no path construction among storing nodes and the sink is necessary, since the sink will directly contact the storing nodes. For the special case where $s = \sqrt{n}$ (i.e. $s \neq n$), we get $r(s) \approx \frac{n}{n-s} \approx \sqrt{n}$. This means that for relatively small view sizes, there is a very little chance of getting collisions and that by only contacting $m = \sqrt{n}$ storing nodes a sink can get a representative view of the whole data in the network (i.e., $s * m = \sqrt{n} * \sqrt{n} = n$).

Regarding the assignment of dynamic weights to nodes, such as energy-based weights, it is enough to associate energy-level thresholds with weights and to redo the weights distribution phase each time the node energy falls below the threshold. In this way, the probability of a node to be selected as a storing node would be given by its weight and consequently, by its remaining energy. Note that here, we only discuss how the flexible data dissemination algorithm Supple

can be performed according to the location and weights of nodes. So that, Supple can be adapted to any mechanism of weight assignment. The specification of these mechanisms, however, is not the focus here.

Otherwise, with the construction of the tree, nodes close to the root become hot spots, this can cause battery depletion of these nodes and consequently, tree disconnections. This can be avoided or minimized by the use of specific policies for dynamically modifying the root of the tree, or by the generation and maintenance of multiple and different trees. However, the main goal was to evaluate the main features of the Supple approach. Those kind of improvements are left for future work.

Concerning the complexity, the most important results are the following. The propagation of the weights is done with $\mathcal{O}(n)$ messages. The data dissemination for each node, is done with $\mathcal{O}(r(s) \cdot \log n)$ messages. Thus, the total complexity in term of messages for the data dissemination for all nodes is $\mathcal{O}(n \cdot r(s) \cdot \log n)$, which is then the total number of messages used by Supple.

The table 8.1 summarizes the differences with RaWMS (see [4]) and flooding.

8.4 Performance analysis

8.4.1 Evaluation methodology

Supple is evaluated through simulation using a home-made simulator. Each simulation comprises a dissemination and a gathering phase. In the first phase, each node performs Supple or RaWMS for data dissemination. In the data gathering phase, a mobile sink performs as many visits as necessary to get a representative amount of data of the network, meaning getting n different entries of storing nodes' views.

8.4.1.1 Experimental setup

Four scenarios has been considered, each experiment has been repeated 25 times and the results represent the average value of these experiments. In the first scenario, the same topology and tree are used in all experiments in order to evaluate the effects of the random choices performed by the *Forward_data* routine(cf. Fig. 8.3). In the second scenario, again the same topology is used but different trees are generated for each experiment. By comparing these results to the previous one, it shows how the tree construction impacts the performance of the

protocol. The third scenario compares the Supple protocol to the RaWMS protocol on 25 different topologies. Finally, the forth scenario allows the evaluation of the performance of both protocols in the presence of message losses.

It is worth mentioning that, in order to compare the dissemination capabilities of Supple and RaWMS, no salvation mechanism was added to deal with message losses (e.g., in RaWMS, this mechanism establishes that if a low level acknowledgment is not received for the just sent message, then another random neighbor is chosen by the RW process). Its implementation could improve the performance of both protocol under message losses.

8.4.1.2 Simulation parameters

The simulations involve scenarios with $n = 1,000$ nodes placed at uniformly random locations in a square area. The average number of nodes in the communication range of any node was set to a target average density $d_{avg} = 24$. Only connected topologies were considered, where a binary tree was constructed from the root (cf. section 8.3.1).

For comparison reasons with RaWMS, we set: (1) the size of the target set to $|S| = V = n$ and assign $W(i) = 1$ for all nodes $i \in S$; and (2) nodes view size to $s = \lceil\sqrt{1,000}\rceil = 32$ entries and consequently, $r(s) = \lceil\sqrt{1,000}\rceil = 32$ (cf. Section 8.3.3.1). As a buffer management policy, we consider a *size-based policy*, which removes the oldest entry to make room for the new information in the view. Finally, both protocols, Supple and RaWMS, handle multiple-entry collisions: when adding a new entry to the view, they first check if it is already known, thus no entry is added twice.

8.4.1.3 Evaluation metrics

To evaluate the cost and efficiency of Supple, we evaluate (1) the messages overhead, which counts the amount of transmitted messages by each node (i.e., it is important to ensure that the impact of both the tree and the random choices on the communication are negligible) and (2) the efficiency in data gathering, which is the accumulated amount of collected information after a node is visited by the sink. It is known that, at the worst case, all the data can be gathered by visiting all the nodes in the target set. Nevertheless, we want to know how fast the data can be collected. In fact, by well distributing data amongst storing nodes (and in the case $|S| = n$), Supple allows a mobile sink to perform free trajectories and get a representative amount of information of the whole network by visiting

a small number m, where $m << n$, of storing nodes of the target set. In order to evaluate this property, we consider a mobile sink will cross the network and randomly visit nodes, gathering the data in their views.

8.4.2 Simulated results

The results regarding the previously evoked metrics are presented in the following pages.

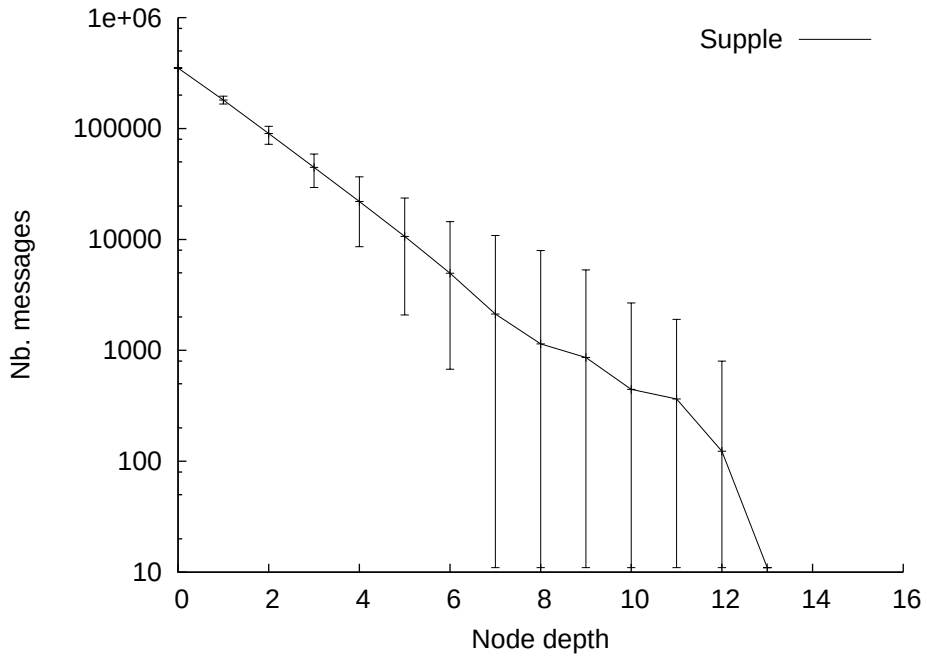


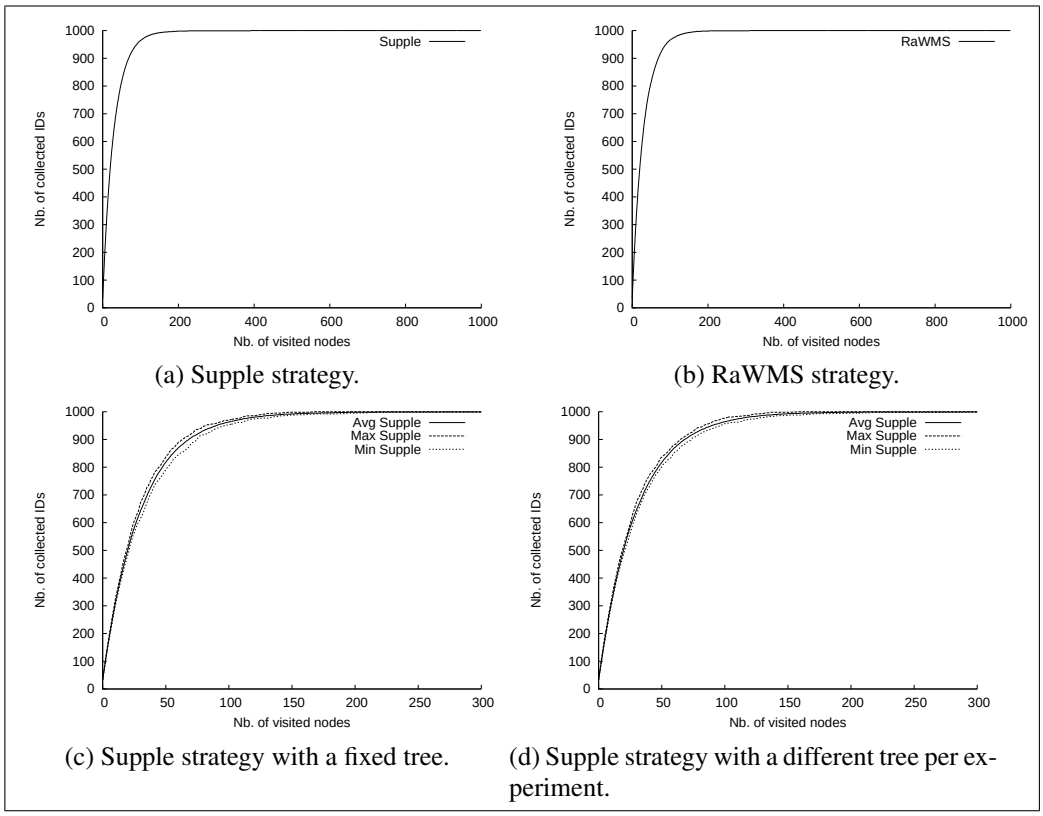
Figure 8.5: Average number of messages sent by a node in Supple as a function of its depth in the tree.

8.4.2.1 Communication overhead

Fig. 8.5 represents the average number of messages sent by a node as a function of its depth in the tree. The number is the average value obtained over 25 experiments, where the tree is the same and only the random choices made in the algorithm are different (cf. Fig. 8.3). We sum all the messages sent by nodes at a certain depth and then divide this number by the number of nodes which are at this particular depth. As expected, the closer the node is to the root in the tree, the more messages it has to send. Thus, as discussed in section 8.3.4, the use of multiples trees and consequently multiples and well distributed roots, could help on the distribution of message overhead amongst nodes in the network.

Additionally, the total number of messages seen at the end of each experiment has been computed. It equals to an average of $3.1967E+06$ with a very small variance for the 25 experiments, which demonstrates the limited influence of the random choices of the protocol. Instead, as shown in [4], RaWMS presents much higher overhead results for a smaller network: up to $5E+03$ messages

for a network size of 800 nodes. When different trees are built on each experiment, the number of messages equals to $3.2E+06$ in average and the variance is still small. This shows that the underlying network, the constructed tree, and the random choices performed during Supple deployment do not really impact the behavior of the protocol as long as it is "well balanced" (*i.e.* each non-leaf node has at least two children). In addition, both results confirm the message overhead analysis discussed in section 8.3.4.



(a) Supple strategy.

(b) RaWMS strategy.

(c) Supple strategy with a fixed tree.

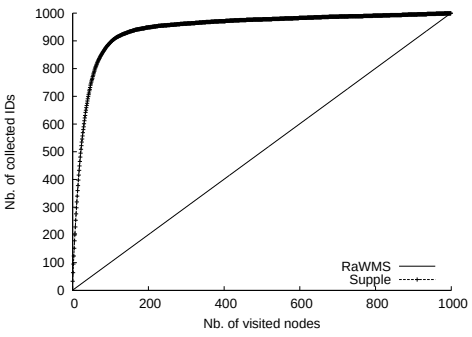
(d) Supple strategy with a different tree per experiment.

8.4.2.2 Efficiency in data gathering

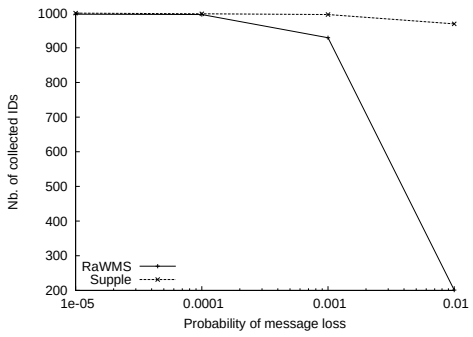
Here, a mobile sink is firstly placed in a random position in the network, it visits the nodes in this position, and then chooses the next node to visit, trying to avoid revisiting an already visited node (as introduced in [34]). When the sink visits a node it gathers this node's data and all the information in its view. This procedure is repeated until the sink has collected all the network information. Fig. 8.6a and 8.6b show the amount of accumulated collected entries (represented in the graph by the number of gotten IDs of the stored data) the mobile sink gathers per visited node. The results indicate that Supple gives similar results to the ones given by RaWMS. In particular, after visiting any $2.3\sqrt{n} \approx 73$ nodes, the sink is able to collect information from about 90% of the nodes, as implied by the analysis in Section 8.3 and [34]. Thus, these results confirm that Supple with an exponential improvement of the number of messages (cf. Table 8.1 in Section 8.3) compared to RaWMS, allows the mobile sink to achieve a high representative view (i.e. 90%) of the whole network data, by only visiting a

relative small number of nodes network, i.e. 73 nodes over $1,000$ (7.3% of nodes).

Additionally, Fig. 8.6c and 8.6d show that random choices as well as the use of different trees at the tree construction phase do not affect the good performance given by Supple in the amount of collected information (i.e. 90% of total data when only 7.3% of network nodes are visited).



(a) Under 5% of loss rate.



(b) After visiting 200 nodes and under different loss rates.

8.4.2.3 Loss resilience

Fig. 8.6a shows the performance in data gathering of both protocols, when the probability of loosing a message is 5%. The performance of Supple is not affected by the loss rate, whereas RaWMS is unable to tolerate this low level of failures. This is clearly understandable since the RaWMS protocol exchanges exponentially many more messages than Supple. The probability for a message to be delivered in the RaWMS protocol with 5% of messages loss is $0.95^{1998} \simeq 3.10286848 \times 10^{-45}$, which is close enough to zero to result in no message reception. In particular, the effect of messages losses in the RaWMS efficiency is only reduced when unicast packet retransmissions and the salvation mechanism is implemented, as shown in results in [4].

Finally, Fig. 8.6b shows the averaged number of data the sink has gathered information after having visited 200 nodes, under different loss rates. We can see that when the message loss percentage is higher than $1‰$, the performance of the RaWMS protocol decreases rapidly, while the Supple protocol still keeps good performances. This is again explained by the fact that RaWMS exchanges exponentially many more messages than Supple.

8.4.3 Conclusion

Supple provides *an exponential improvement of the number of messages used*, when compared to RaWMS. The choice to use a tree, however, introduces a high overhead of messages transmission to the root and its vincinity. A solution to this consists in creating multiple trees with different roots and load balancing the data dissemination on the different trees, alleviating the communications requirements imposed to a unique root. Such solution can be easily implemented by randomly selecting uniformly distributed nodes in the network for initiating the tree construction.

Additionally, the simulations demonstrate that Supple behaves as predicted by the theoretical analysis: a very high proportion of n data can be distributed in a way that, it is still possible to gather most of the network data by only visiting a small portion $m = 2.3\sqrt{n}$ of the target set, where $m << n$. Regarding efficiency in data gathering, Supple provides *the same quality of data dissemination as the RaWMS protocol, but with more flexibility,* since the storing nodes may be selected following any criterion of distribution. The simulations also illustrated that, due to its relatively small number of message transmissions when compared to RaWMS, *Supple can tolerate a much higher failures rate without requesting additional link reliability or extra salvation mechanisms*.

CHAPTER 9

TEST FOR LARGE SCALE APPLICATIONS

This chapter introduces a testbed for large-scale applications called V-DS. This testbed is an emulator and offers a complete control over the experimental conditions when stressing an application.

V-DS introduces virtualization of all the hardware of the parallel machine, and of the network conditions. It provides the experimenter with a tool to design a complex and realistic failure scenario, over arbitrary network topologies. To the best of my knowledge, this is one of the first systems to virtualize all the components of a parallel machine, and provide a network emulation that enables experimenters to study low-level network protocols and their interactions with failures.

9.1 V-DS Platform Description

The V-DS emulation environment is made up of two distinct components. First, the virtualization of the hardware through the use of virtual machines and then a BSD kernel module for the low-level network virtualization enabling the emulation of grids on clusters. Each component will be described in the following sections.

9.1.1 Virtualization Environment for Large-scale Distributed Systems

V-DS virtualizes distributed systems entities, at both operating and network level. This is done by providing each virtual node with its proper and confined operating system and execution environment.

V-DS virtualizes a full software environment for every distributed system node. It allows the accurate folding of a distributed system up to 100 times larger than the experimental infrastructure [82], typically a cluster.

V-DS supports three key requirements:

- Scalability: In order to provide insights on large-scale distributed systems, V-DS supports the folding of distributed systems. Indeed, thanks to Xen characteristics, it is possible to incorporate a large number of virtual machines on a single physical machine with a negligible overhead [5].

- Accuracy: In order to obtain accurate behavior of a large-scale distributed system several constraints on the virtual machines (VMs) are needed. First the CPU must be fairly shared between VMs, then each VM must be isolated from the others, lastly the performance degradation of a VM must evolve linearly with the growth of the number of VMs. Using Xen allows V-DS to ensure all these requirements (see for example [82]).

- Adaptivity: the platform provides a custom and optimized view of the physical infrastructure used. For instance, it is possible to support different operating systems, and even different versions of the same operating system.

V-DS is based on the Xen [5] virtualization tool in version 3.2. Xen gets interesting configuration capabilities, particularly at the network level which is fundamental in the injection of network topologies. Compared to other virtualization technologies it has been demonstrated [82] that Xen offers better results.

Figure 9.1 shows the general architecture of V-DS. Here, m physical machines called $PM - i$ running a Xen system are hosting n virtual machines named $VM - i - j$ with i the index of the physical machine hosting that virtual machine and j the index of the virtual machine. Thus, there are $n * m$ virtual machines (VM). All communications between these VM are routed to FreeBSD machines to 1) prevent them from communicating directly through the internal network if they are on the same physical machine, 2) add network topologies between VM.

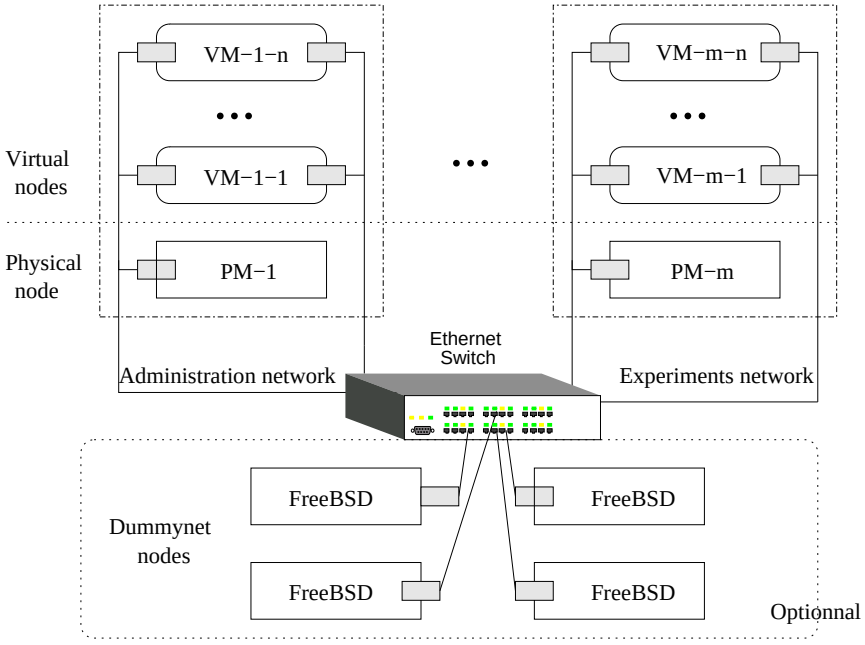


Figure 9.1: Overview of the architecture of V-DS.

9.1.2 Low-level Network Virtualization

One of the main advantages of the V-DS platform is that it also uses virtualization techniques for emulating the network. This allows the experimenter to emulate any kind of topology with various values for latency and bandwith on a cluster. For instance, it is possible to run, in this framework, grid applications on clusters.

For the purpose of emulating the network, we use FreeBSD machines. Using BSD machines to virtualize the network is crucial to a reliable accurate network simulation, since BSD contains several very efficient tools to manipulate packages like ipfw[1] or Dummynet[86]. With these packages it is possible to insert failures (like packet dropping) in the network very easily. The platform is then able to inject realistic failures at the machine and network level.

There are three networks joining the virtual machines. The first one is a classic ethernet network. Each virtual machine has its own ethernet card. The second one uses Myrinet cards and provides very fast links between the nodes. The last

[1] `http://www.freebsd.org/doc/en/books/handbook/firewalls-ipfw.html`

one offers layer 2 virtualization using the EtherIP protocol [52]. EtherIP bridges are set between any virtual machine and the corresponding BSD machine.

To set up the bridges, we use a topology file given by the user. The format we use for the topology is the dot format [2] which is easy to manipulate and to write. There are two different types of nodes for the topology. The Xen nodes representing the virtual machines and the BSD nodes representing routers. The dot language being very simple it is easy to generate well-known topologies such as rings, cliques, etc. As the language is well-spread, there also exist graphical tools to design specific topologies.

Using the topology file, we generate the routing table of each BSD machine. The routing is made through a kernel module. More precisely it is a netgraph[3] node called "ip_switch". This node works with ipfw, allowing the user to filter packets and to redirect them into a netgraph node. Here we filter all EtherIP packets.

These packets are examined by the module who stores its routing table in a kernel hash table. After modifying the IP header of the packet to correctly route it to the next hop, the packet is put back on the network. The packet may also enter a Dummynet rule before or after being rerouted. The module can also deal with ARP (Address Resolution Protocol) requests in which case it will forward the request to all its neighbors.

9.2 Experiments

In this section I present the experiments performed in order to assess the performances and functionalities of the virtualization framework. All these experiments were done on Grid'5000 [16]. Grid'5000 is a computer science project dedicated to the study of grids, featuring 13 clusters, each with 58 to 342 PCs, connected by the Renater French Education and Research Network. For these experiments we used a 312-node homogeneous cluster composed of AMD Opteron 248 (2.2 GHz/1MB L2 cache) bi-processors running at 2GHz. Each node feature 20GB of swap and SATA hard drive. Nodes were interconnected by a Gigabit Ethernet switch. All these experiments were performed using a folding ratio of 10 (e.g. each physical node runs 10 virtual machines).

All the following experiments ran under Xen version 3-2, with Linux-2.6.18.8 for the Physical and Virtual Computing nodes, and BSD version 7-0PRERELASE

[2] `http://www.graphviz.org/doc/info/lang.html`

[3] `http://people.freebsd.org/~julian/netgraph.html`

Requested value	Measured value - w/o NE	Measured value - with NE
250 Mbps	241.5 Mbps	235 Mbps
25 Mbps	24.54 Mbps	23.30 Mbps
2,5 Mbps	2.44 Mbps	2.34 Mbps
256 Kbps	235.5 Kbps	235.5 Kbps

(a) Bandwidth restraint

Requested value	Measured value - w/o NE	Measured value - with NE
10 ms	10.1 ms	10.2 ms
50 ms	52.2 ms	52.1 ms
100 ms	100.2 ms	100.4 ms
500 ms	500.4 ms	500.8 ms

(b) Latency restraint

Table 9.1: Respect of network restraint conforming measures

for the network emulation. Since we were embedding a Java virtual machine on the Xen virtual machines we needed it to be light. We chose the 1.5.0_10-eval version that fulfilled our needs and the space requirements.

9.2.1 Impact of the Low-Level Network Emulation

We first measured the impact of the network emulation of V-DS on the network bandwidth and latency, using the netperf tool. To do this, we used two version of V-DS: with network emulation at the high level only (when packets are slowed down by the router, but not encapsulated in an IP over ethernet frame), and with low-level network emulation, as described in section 9.1.

The experimental setup consisted in three physical machines: one running the BSD router, the other two running one virtual machine each. We configured the BSD router to introduce restraints on the network, either using low-level emulation with ethernet over IP, or without low-level emulation.

The results are summed up in table 9.1. The requested value represents the restraint imposed by the experiment. In this table low-level network emulation is denoted as NE.

Regarding the bandwidth, the obtained values are very close to the requested ones, the difference being around 3% without the low-level network emulation. This corresponds to the time spent in the crossing of the virtual layer. Netperf

tests are realised at the TCP level, implying that part of the bandwidth is used for the TCP protocol.

When adding the low-level network emulation the bandwidth drops again for another 3%. This could be explained by the encapsulation needed by the etherip protocol for a full network emulation. There is no significant difference regarding the latency measures with the low-level emulation.

9.2.2 TCP Broken Connection Detection Mechanism

In these sets of experiments, We stress the broken connection detection mechanism implemented in the TCP stack. Many applications rely on TCP detection mechanism to detect failures and implement their own fault-tolerance strategy, thus the efficiency of TCP failure detection has a significant impact on the efficiency of these applications. The failure detection mechanism of TCP relies on heartbeats, under the so-called pull model [10]: one peer sends a heartbeat to the other peer, and starts a timer; when a peer receives a heartbeat, it will send an acknowledgement back; if the acknowledge returns before the expiration of the timer, the sending peer assumes that the receiving peer is alive; if the timer expires before the reception of the acknowledge, the connection is broken.

This mechanism is controlled at the user level through BSD socket parameters: *SO_KEEPALIVE* enables the failure detection mechanism, which is tuned through *tcp_keepalive_time*, *tcp_keepalive_probes*, and *tcp_keepalive_intvl*. First, *tcp_keepalive_time* defines how long a socket can be without traffic before beginning the heartbeat protocol; *tcp_keepalive_probes* defines the number of heartbeats that can be lost on a socket before the connection is considered broken; *tcp_keepalive_intvl* defines the maximum time to wait before considering that a heartbeat has been lost.

To stress the failure detection mechanism of TCP, we designed three simple synthetic benchmarks. They all assume a single pair of client/server processes connected through TCP BSD sockets. In the first benchmark (*Send*), the client sends messages continuously to the server without waiting for any answer. In the second benchmark (*Recv*), the client awaits for a message from the server. In the third (*Dialog*), the client and server are alternatively sending and receiving messages to/from each other.

In all those experiments the server is killed or destroyed right after the connection is established and we measure the elapsed time before the client realizes the connection has been broken. We set the *tcp_keepalive_time* to 30 minutes, the *tcp_keepalive_probes* to 9 and the *tcp_keepalive_intvl* to 75 seconds, which

are the default on linux machines (except for the keepalive time, which was reduced to lower the duration of the experiments). As a consequence, when a machine crashes we expect the other side to notice the event within a period of approximately 41 minutes.

We then have two sets of experiments, one where the server process is killed and one where the machine hosting the process is destroyed. Each set of experiments includes the three benchmarks in both Java and ANSI C. Every benchmark is run twice, once where the SO_KEEPALIVE variable is on and once where it's off. The results are summed up in table 9.2.

Failure Injection Method	Language	Socket Option	Send	Receive	Dialog
Kill	C	-	0.2s	0.3s	0.2s
Kill	Java	-	N/A	N/A	N/A
Destroy	C	SO_KEEPALIVE	17min	41min	15min30s
Destroy	Java	SO_KEEPALIVE	∞	41min	15min30s
Destroy	C		17min	∞	15min30s
Destroy	Java		∞	∞	15min30s

Table 9.2: TCP failure detection times

In this table, the value ∞ means that after a long enough amount of time (several hours) exceeding significantly the expected time of the failure detection (41 minutes) the active computer has still not noticed that the connection has been broken. The N/A value means that the language or the system does not notify errors even if it detects a broken link. In the case of failure injection with the Kill method, the socket option SO_KEEPALIVE has no effect on the results.

When using the Kill failure injection method, one can see that the Linux operating system detects the failure at the other end almost instantaneously. This is due to the underlying TCP/IP protocol: the process is killed, but the operating system continues to work, so it can send the RST packet to the living peer, which will catch it and notify the process of a "failure". For the Java virtual machine, the socket is also notified as closed, but the language does not notify this closure as a failure: the code also has to check continuously for the status of the Input and Output streams, in order to detect that a stream was unexpectedly closed. In

the JVM implementation used, no exceptions were raised when sending on such a stream, and receptions gave null messages.

On the contrary, when using a more realistic destroy mechanism, the operating system of the "dead" peer is also destroyed. So, no mechanism sends a message to the living peer to notify of this crash. The living peer must rely on its own actions to detect failures, which is a more realistic behavior. We distinguish between the two cases when the SO_KEEPALIVE option is either on or not on the socket. In native Linux applications (ANSI C programs), the failure is always detected when the SO_KEEPALIVE option is on. TCP also uses the communications induced by normal traffic to detect a potential failure, that is why the detection time is lower for the Dialog and Send benchmarks.

In the case of the Recv benchmark, the living peer does not introduce communication in the network, so the system has to rely on the heartbeat procedure, which uses conservative values to detect the failure with a low chance of false positives, and a small perturbation of the network.

It is clear from these experiments that crash injection through complete destruction of the virtual machine, including the operating system, exhibit more accurate behavior than the simple destruction of a process, even using a forced kill method, because the underlying operating system will clean up the allocated resources, including the network resources.

9.2.3 Stress of Fault-Tolerant Applications

In order to evaluate the platform capabilities to inject failures, We stressed FreePastry which is an open-source implementation in Java of Pastry [88, 19] intended for deployment on the Internet. Pastry is a fault-tolerant peer-to-peer protocol implementing distributed hash-tables. In Pastry every node has a unique identifier which is 128 bits long.This identifier is used to position the node on a 2^{128}-place oriented ring. A key is associated to any data, using a hash function, and each process of identifier $id < id'$ (where id' is the identifier of the next process on the ring) holds all data with key k such that $id \leq k < id'$. Then, by comparing the process identifiers and data keys, any process can route any message to a specific data. Shortcuts between nodes (called fingers in Pastry) are established to ensure logarithmic time to locate a node holding any data from any other node.

When a node is joining an existing ring, it gets a node id and initializes its leaf set and routing table by finding a "close" node according to a proximity metric. Then it asks this node to route a special message with its identifier as a

key. The node at the end of the road is the one with the closest identifier and then the new node takes its leaf set and its routing table is updated with information gathered along the road. The new node will then send messages in the pastry network to update the routing table of all processes it should be connected to.

Pastry manages nodes failures as nodes departures without notification. In order to handle this kind of situation "neighbors" (nodes which are in each others leaf set) exchange keepalive messages. If a node is still not responding after a period T, it is declared failed and everyone in its leaf set is informed. The routing table of all processes that the departing process was connected to are then updated. This update procedure can take some time and is run during the whole life of the distributed hash table. At some point in time, the routes stop changing (they are stabilized), but the maintaining procedures for these routes continue to execute.

In order to validate the platform we look at three things. First we evaluate the average time for the system to stabilize itself after all the peers had joined the network. Then we evaluate the average time needed for every node to know that a node was shut down or killed. In the first case we only kill a java process and in the second we "destroy" the virtual machine which is hosting the process.

The experiments go as follows. The first virtual machine (called the bootstrap node) creates a new ring and then every other virtual machine connects to it. We ask every node for its routing table every 200ms and log it whenever it changes together with a time stamp.

In order not to overwhelm the bootstrap node, machines are launched by groups of tens separated by a 1 second interval. The results for the first experiment are presented in Fig. 9.2 below.

It can be seen that for even small rings, composed of as few as 50 machines out of a possible 2^{128}, the time for the system to stabilize is huge (over 5 hours). This time increases with the numbers of machines and can still be over 18h for a ring as small as 400 machines.

To reduce the duration of the experiments, We use the fact that a majority of changes in the routing tables are made in the first few seconds of initialization. It appears that after only 100s more than 50% of the changes have been made. Thus we do not wait for the whole system to be stabilized before injecting the first failure, but we only wait for the whole system to have made enough changes in the routing tables and for it to be in a relatively steady state. The first failure is injected 45min after the beginning of the experiment.

We call D-node the node suppressed from the ring, either by killing the process or destroying the machine. After suppressing the D-node we wait for 20

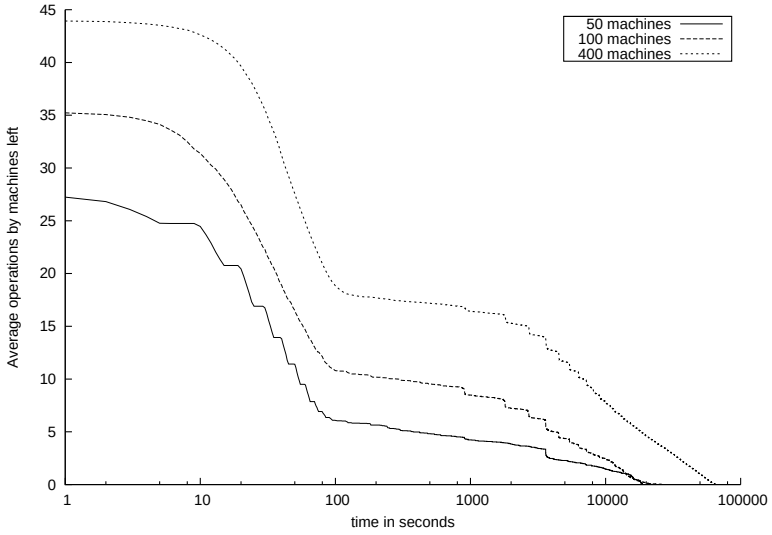


Figure 9.2: Average number of changes left by machines

min for the nodes to update their routing tables. After this period we collect the routing tables and look for those which include the D-node. In those particular tables we search for the update that will make the D-node disappear from the routing table.

Figure 9.3 presents the cases when we "destroy" the virtual host of the process, and when we kill the process. Each dot in this figure represents the update of the routing table of process y, at a time x, concerning the D-node. The circles represent the modifications before the failure is injected, thus modifications due to the normal stabilization of FreePastry. The squares represent the modifications after the injection of the failure for the D-node in the case of process kill, and the triangles in the case of virtual host destruction. The vertical line represents the date of the failure injection at the D-node (45 minutes after the beginning).

The set of routing tables that include the D-node consists of 578 nodes over several experiments. In this set many nodes delete the D-node of their routing table before it is suppressed. As it can be seen on the figure, all these nodes do it

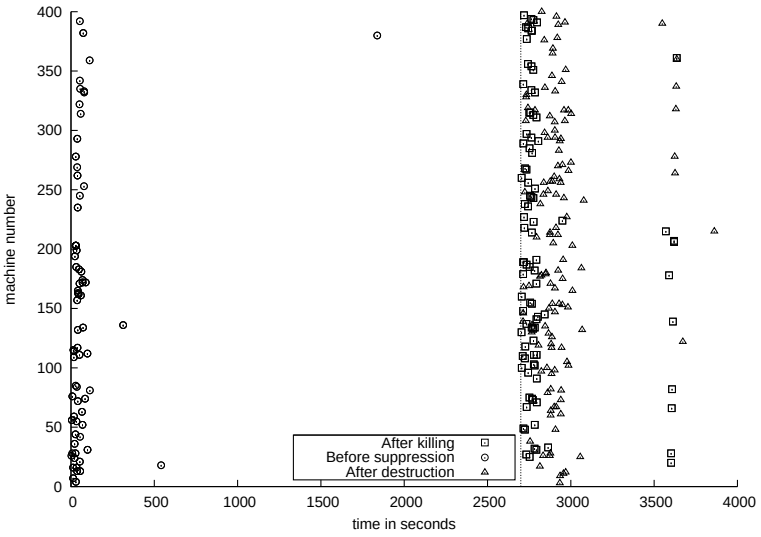


Figure 9.3: D-node deletion time

very early in the stabilization and therefore we can consider that every node that deletes the D-node from its table after the suppression time does it thanks to the failure detection component of Pastry.

Since the routing table maintenance is done lazily in Pastry [20], it is natural that not every node updates its routing table, since in the experiments no messages are exchanged.

When we only kill the pastry process to suppress the D-node after 45 minutes it is possible to see on figure 9.3 that a lot of nodes react in a very short period of time to the suppression of the D-node. Comparing the points distributions for Kill and Destruction, one can see that nodes detect the failure in a shorter period of time in the case of kill than in the case of destruction. Since behaviors in the two cases is different one should consider that "destroying" a machine is more accurate since the stressed application must rely on its own failure detection mechanism, and the behavior of this application may be influenced by the asynchronism and the timings of the failure detection mechanism used. The figure also demonstrates that the active failure detection mechanism of FreePastry

is effective and the distributed hash table is able to stabilize even with accurate failure injection.

9.3 Conclusion

The platform V-DS helps the developer to eliminate bugs during the development of his applications. He can stress his application at every level since he possess a complete control over the environment.

This platform offers a total virtualization of machines and networks at both high and low level. This allows the user to inject realistic faults into the system to test its reaction and the fault-tolerance of the application.

Since the environment is totally harnessed, it is possible to log almost everything that happens and to use this information as input for verification tools such as APMC [48] or together with a fault injection tool like Fail [51] to reproduce scenarios precisely.

Part III

Conclusion

CHAPTER 10

CONCLUSION

This thesis resolves several large-scale related issues. The large-scale systems addressed are various and cover huge data collection like the Web, sensor networks with hundreds of thousands of nodes or clusters and grids applications. The results are methods to fight social spam and Webspam, a new dissemination data protocol for sensor networks and a testbed for large-scale applications.

10.1 Social spam

Chapter 4 presented a voting scheme for social news websites that prevent spammers' content to reach the front page. The method consists in the application of statistical filters and a new algorithm to detect communities. Its implementation through a website was clearly a success, assessing the notions used in its development.

10.2 Webspam demotion

In chapter 5, I presented a new method to compute a fairer ranking on search engines. It is based on a clustering of the graph as a preprocessing step and rank computed with only extra-cluster contributors. Fast and simple clustering

techniques that uses only local information where introduced and one of them can statistically make a difference between spam and nonspam pages.

It is now of interest to have a better look at the clustering produced by those methods and if they are actually good approximate clustering methods.

10.3 Webspam detection

Another approach to minimize the influence of Webspam was developed with the technique presented in chapter 6. It consists in identifying pages whose ranking is ameliorated by Webspam. The method is simple, it uses random walks and their statistical projections introduced in [31].

Efforts should be made to define criteria to identify suspicious to shorten the running time of the algorithm.

10.4 Supple

A new data dissemination scheme for wireless sensor networks is presented in chapter 8. This scheme, called Supple, exchange much less messages than the previous ones to attain the same performances. By using a tree topology over the network that can be created with only local knowledge it is possible to use driven random walks without distorting the distribution over the storing nodes.

10.5 V-ds

In chapter 9, I presented a new testbed for large-scale applications that allows the user to inject accurate failures to test his applications. It virtualizes both the machines and the network to offer a complete control over the environment to the user. The validity of this platform was tested using network tools and a fault-tolerant implementation of a distributed hash table called FreePastry.

BIBLIOGRAPHY

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

[2] R. Andersen, C. Borgs, J. Chayes, J. Hopcroft, K. Jain, V. Mirrokni, and S. Teng. Robust pagerank and locally computable spam detection features. In *AIRWeb '08: Proceedings of the 4th international workshop on Adversarial information retrieval on the web*, pages 69–76, New York, NY, USA, 2008. ACM.

[3] K. Arrow. A difficulty in the concept of social welfare. *The Journal of Political Economy*, 58(4):328–346, 1950.

[4] Z. Bar-Yossef, R. Friedman, and G. Kliot. RaWMS - Random Walk based lightweight Membership Service for wireless ad hoc networks. *ACM Trans. Comput. Syst.*, 26(2):1–66, June 2008.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In M. L. Scott and L. L. Peterson, editors, *SOSP*, pages 164–177. ACM, 2003.

[6] S. Basagni, A. Carosi, E. Melachrinoudis, C. Petrioli, and Z. M. Wang. Controlled sink mobility for prolonging wireless sensor networks lifetime. *Wireless Networks Journal (WINET)*, 14(6):831–858, Dec. 2008.

[7] W. H. Bell, D. G. Camerona, L. Capozza, A. P. Millar, K. t Stockinger, and F. Zini. Optorsim - a grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications,*, 17(4), 2003.

[8] A. A. Benczur, K. Csalogany, T. Sarlos, M. Uher, and M. Uher. Spamrank - fully automatic link spam detection. In *In Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb*, 2005.

[9] T. Berners-Lee. The world-wide web. *Computer Networks and ISDN Systems*, 25(4-5):454–459, 1992.

[10] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 354–363, Washington, DC, USA, 2002. IEEE Computer Society.

[11] J. Bian, Y. Liu, E. Agichtein, and H. Zha. A few bad votes too many?: towards robust ranking in social media. In *Proceedings of the 4th international workshop on Adversarial information retrieval on the web*, pages 53–60, New York, NY, USA, 2008. ACM.

[12] P. Boldi and S. Vigna. The webgraph framework I: Compression techniques. In *In Proc. of the Thirteenth International World Wide Web Conference*, pages 595–601. ACM Press, 2003.

[13] P. Brémaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*. springer verlag, 1999.

[14] E. Buchmann and K. Böhm. How to run experiments with large peer-to-peer data structures. *Parallel and Distributed Processing Symposium, International*, 1:27b, 2004.

[15] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource manageme nt and scheduling for grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14(13-15), 2002.

[16] F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jegou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, and O. Richard. Grid'5000: a large scale, reconfigurable, controlable and monitorable Grid platform. In *SC'05: Proc. The 6th IEEE/ACM International Workshop on Grid Computing Grid'2005*, pages 99–106, Seattle, USA, November 13-14 2005. IEEE/ACM.

[17] A. Carneiro-Viana, T. Hérault, T. Largillier, S. Peyronnet, and F. Zaïdi. Supple: A flexible probabilistic data dissemination protocol for wireless sensor networks. In *Proceedings of the Thirteenth ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. Sheridan Printing, 2010.

[18] H. Casanova. Simgrid: a toolkit for the simulation of application scheduling. In *Proceedings of the IEEE International Symposium on Cluster Computing and the G rid (CCGrid'01),Brisbane, Australia*, pages 430–437, may 2001.

[19] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Security for structured peer-to-peer overlay networks. In *5th Symposium on Operating Systems Design and Implementaion (OSDI'02)*, Dec. 2002.

[20] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Topology-aware routing in structured peer-to-peer overlay networks. 2003.

[21] I. Chatzigiannakis, A. Kinalis, and S. Nikoletseas. Efficient data propagation strategies in wireless sensor networks using a single mobile sink. *Computer Communications*, 31(5):896–914, 2008.

[22] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[23] Y.-j. Chung, M. Toyoda, and M. Kitsuregawa. A study of link farm distribution and evolution using a time series of web snapshots. In *AIRWeb '09: Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web*, pages 9–16, New York, NY, USA, 2009. ACM.

[24] D. Cosley, S. K. Lam, I. Albert, J. A. Konstan, and J. Riedl. Is seeing believing?: how recommender system interfaces affect users' opinions. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 585–592, New York, NY, USA, 2003. ACM.

[25] D. Cruller, D. Estrin, and M. Srivastava. Overview of sensor networks. *Computer*, 37(8):41–49, 2004.

[26] C. de Kerchove, L. Ninove, and P. Van Dooren. Maximizing PageRank via outlinks. *Linear Algebra and its Applications*, 429(5-6):1254–1276, 2008.

[27] C. Dumitrescu and I. Foster. Gangsim: A simulator for grid scheduling studies. In *Proceedings of the IEEE International Symposium on Cluster Computing and the G rid (CCGrid'05), Cardiff, UK*, may 2005.

[28] J. Eriksson, M. Faloutsos, and S. Krishnamurthy. Peernet: Pushing peer-to-peer down the stack. *Proceedings of the International Workshop on Peer-to-Peer Systems*, Feb. 2003.

[29] J. Eriksson, M. Faloutsos, and S. Krishnamurthy. Scalable ad hoc routing: The case for dynamic addressing. In *Proccedings of IEEE Infocom*, Mar. 2004.

[30] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol–HTTP/1.1, 1999.

[31] E. Fischer, F. Magniez, and M. de Rougemont. Approximate satisfiability and equivalence. *Logic in Computer Science, Symposium on*, 0:421–430, 2006.

[32] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Trans. Netw.*, 9(4):392–403, 2001.

[33] I. Foster. What is the grid? a three point checklist. June 2002.

[34] R. Friedman, G. Kliot, and C. Avin. Probabilistic quorum systems in wireless ad hoc networks. In *Proceedings of the 9th IEEE Conference on Dependable Systems and Networks (DSN)*, June 2008.

[35] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *PROC.NATL.ACAD.SCI.USA*, 99:7821, 2002.

[36] P. Gupta and P. Kumar. Critical power for asymptotic connectivity in wireless networks. In *Proceedings of Stochastic Analysis, control, optimization and applications*, pages 547–566, 1998.

[37] Z. Gyongyi, P. Berkhin, H. Garcia-Molina, and J. Pedersen. Link spam detection based on mass estimation. In *VLDB '06: Proceedings of the*

32nd international conference on Very large data bases, pages 439–450. VLDB Endowment, 2006.

[38] Z. Gyöngyi and H. Garcia-Molina. Web spam taxonomy. *Adversarial Information Retrieval on the Web*, 2005.

[39] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 576–587. VLDB Endowment, 2004.

[40] A. Haeberlen, A. Mislove, A. Post, and P. Druschel. Fallacies in evaluating decentralized systems. In *In Proceedings of IPTPS*, 2006.

[41] C. Hailing, M. Yong, L. Tianpu, L. Wei, and Z. Ze. Overview of Wireless Sensor Networks. *Journal of Computer Research and Development*, 1, 2005.

[42] E. B. Hamida and G. Chelius. A line-based data dissemination protocol for wireless sensor networks with mobile sink. In *Proceedings of IEEE International Conference on Communications (ICC)*, May 2008.

[43] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocols for wireless microsensor networks. In *Proceedings of Hawaaian International Conference on Systems Science*, Jan. 2000.

[44] W. Heinzelman, J. Kulik, and H. Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *Proccedings of ACM Mobicom*, pages 174–185, Seattle, WA, Aug. 1999.

[45] J. M. Hellerstein, W. Hong, S. Madden, and K. Stanek. Beyond average: Toward sophisticated sensing with queries. Apr. 2003.

[46] T. Hérault, T. Largillier, S. Peyronnet, B. Quétier, F. Cappello, and M. Jan. Emulation platform for high accuracy failure injection in grids. In W. Gentzsch, L. Grandinetti, and G. Joubert, editors, *Advances in Parallel Computing*, volume 18, pages 127–140. IOS Press, 2009.

[47] T. Hérault, T. Largillier, S. Peyronnet, B. Quétier, F. Cappello, and M. Jan. High accuracy failure injection in parallel and distributed systems using

virtualization. In *Proceedings of the 6th ACM conference on Computing frontiers*, pages 193–196. ACM, 2009.

[48] T. Herault, R. Lassaigne, and S. Peyronnet. APMC 3.0: Approximate verification of discrete and continuous time Markov chains. In *Proceedings of the Third International Conference on the Quantitative Evaluation of Systems*. IEEE Computer Society, 2006.

[49] P. Heymann, G. Koutrika, and H. Garcia-Molina. Fighting spam on social web sites: A survey of approaches and future challenges. *IEEE Internet Computing*, 11(6):36–45, 2007.

[50] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the emulab network testbed. In *Proceedings of the 2008 USENIX Annual Technical Conference*, Boston, MA, June 2008.

[51] W. Hoarau, S. Tixeuil, and F. Vauchelles. Fault injection in distributed java applications. Technical Report 1420, Laboratoire de Recherche en Informatique, Université Paris Sud, October 2005.

[52] R. Housley and S. Hollenbeck. EtherIP: Tunneling Ethernet Frames in IP Datagrams. RFC 3378 (Informational), Sept. 2002.

[53] N. Immorlica, K. Jain, M. Mahdian, and K. Talwar. Click fraud resistant methods for learning clickthrough rates. In *Proceedings of the 1st Workshop on Internet and Network Economics 3828. Springer LNCS (2005) 34–45*.

[54] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of ACM Mobicom*, pages 56–67, Boston, MA, Aug. 2000.

[55] B. J. Jansen. Adversarial information retrieval aspects of sponsored search. In *Proceeings of the Second International Workshop on Adversarial Information Retrieval on the Web*, pages 33–36, 2006.

[56] A. Kinalis and S. Nikoletseas. Adaptive redundancy for data propagation exploiting dynamic sensory mobility. In *Proceedings of 11th ACM MSWiM*, pages 149–156, Oct. 2008.

[57] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[58] V. Krishnan and R. Raj. Web Spam Detection with Anti-Trust Rank. *Proceedings of the Second International Workshop on Adversarial Information Retrieval on the Web*, pages 37–40, 2006.

[59] S. K. Lam and J. Riedl. Shilling recommender systems for fun and profit. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 393–402, New York, NY, USA, 2004. ACM.

[60] T. Largillier, G. Peyronnet, and S. Peyronnet. SpotRank: a robust voting system for social news websites. In *Proceedings of the 4th workshop on Information credibility*, pages 59–66. ACM, 2010.

[61] T. Largillier and S. Peyronnet. Lightweight clustering methods for webspam demotion. In *Proceedings of the Ninth international Conference on Web Intelligence*. IEEE Press, 2010.

[62] T. Largillier and S. Peyronnet. Using patterns in the behavior of the random surfer to detect webspam beneficiaries. In *Proceedings of the First international Symposium on Web Intelligence Systems & Services*. LNCS, 2010.

[63] K. Lerman. Social information processing in news aggregation. *IEEE Internet Computing: special issue on Social Search*, 11(6):16–28, November 2007.

[64] K. Lerman. User participation in social media: Digg study. In *Proceedings of the 2007 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, pages 255–258, Washington, DC, USA, 2007. IEEE Computer Society.

[65] K. Lerman. Dynamics of a collaborative rating system. In *Advances in Web Mining and Web Usage Analysis: 9th International Workshop on Knowledge Discovery on the Web, WebKDD 2007, and 1st International Workshop on Social Networks Analysis, SNA-KDD 2007, San Jose, CA, USA, August 12-15, 2007. Revised Papers*, pages 77–96, Berlin, Heidelberg, 2009. Springer-Verlag.

[66] K. Lerman and A. Galstyan. Analysis of social voting patterns on digg. In *WOSP '08: Proceedings of the first workshop on Online social networks*, pages 7–12, New York, NY, USA, 2008. ACM.

[67] X. Liu, Q. Huang, and Y. Zhang. Balancing push and pull for efficient information discovery in large-scale sensor networks. *IEEE Trans. on Mobile Computing*, 6(3):241–251, 2007.

[68] J. Luo and J. P. Hubaux. Joint mobility and routing for lifetime elongation in wireless sensor networks. In *Proceedings of the IEEE INFOCOM 2005*, Miami, FL, USA, Mar. 2005.

[69] J. Luo, J. Panchard, M. Piorkowski, M. Grossglauser, and J. P. Hubaux. Mobiroute: Routing towards a mobile sink for improving lifetime in sensor networks. In *Proceedings of the 2nd IEEE/ACM International Conference on Distributed Computing in Sensor Systems (DCOSS06)*, pages 480–497, San Francisco, CA, USA, June 2006.

[70] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation Service for Ad-Hoc Sensor Networks. *Proccedings of ACM SIGOPS Operating Systems Review*, 36, 2002.

[71] F. Marcelloni and M. Vecchio. A simple algorithm for data compression in wireless sensor networks. *Communications Letters, IEEE*, 12(6):411–413, June 2008.

[72] A. Metwally, D. Agrawal, A. E. Abbad, and Q. Zheng. On hit inflation techniques and detection in streams of web advertising networks. *Distributed Computing Systems, International Conference on*, 0:52, 2007.

[73] R. Motwani and P. Raghavan. Randomized algorithms. *ACM Comput. Surv.*, 28(1):57–63, 1996.

[74] R. Niederberger. DEISA: Motivations, strategies, technologies. In *Proc. of the Int. Supercomputer Conference (ISC'04)*, 2004.

[75] J. Norris. *Markov chains*. Cambridge Univ Pr, 1998.

[76] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting spam web pages through content analysis. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 83–92, New York, NY, USA, 2006. ACM.

[77] L. Nussbaum and O. Richard. Lightweight emulation to study peer-to-peer systems. *Concurr. Comput. : Pract. Exper.*, 20(6):735–749, 2008.

[78] M. O'Mahony, N. Hurley, N. Kushmerick, and G. Silvestre. Collaborative recommendation: A robustness analysis. *ACM Trans. Internet Technol.*, 4(4):344–377, 2004.

[79] T. O'reilly. What is web 2.0. *Design patterns and business models for the next generation of software*, 30:2005, 2005.

[80] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[81] A. Plangprasopchok and K. Lerman. Constructing folksonomies from user-specified relations on flickr. In *Proceedings of the International World Wide Web Conference*, April 2009.

[82] B. Quétier, V. Neri, and F. Cappello. Scalability Comparison of Four Host Virtualization Tools. *Journal of Grid Computing*, 5:83–98, 2006.

[83] F. Radlinski and T. Joachims. Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. In *Conference of the Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1406–1412, 2006.

[84] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 96–108, New York, NY, USA, 2003. ACM.

[85] P. Resnick and R. Sami. The influence limiter: provably manipulation-resistant recommender systems. In *RecSys '07: Proceedings of the 2007 ACM conference on Recommender systems*, pages 25–32, New York, NY, USA, 2007. ACM.

[86] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *SIGCOMM Comput. Commun. Rev.*, 27(1):31–41, 1997.

[87] M. Rosenblum and T. Garfinkel. Virtual machine monitors: Current technology and future trends. *Computer*, 38(5):39–47, 2005.

[88] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Nov. 2001.

[89] G. Saha. Software based fault tolerance: a survey. *Ubiquity*, 7(25):1, 2006.

[90] S. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[91] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin. Data-centric storage in sensornets. *SIGCOMM Comput. Commun. Rev.*, 33(1):137–142, 2003.

[92] K. Swearingen and R. Sinha. Beyond algorithms: An hci perspective on recommender systems. *ACM SIGIR 2001 Workshop on Recommender Systems*, 2001.

[93] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. K. 'c, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and i mplementation*, pages 271–284, New York, NY, USA, 2002. ACM Press.

[94] S. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, May 2000.

[95] A. C. Viana, M. D. Amorim, S. Fdida, and J. F. Rezende. Indirect routing using distributed location information. In *Proceedings of IEEE International Conference on Pervasive Computing and Communications (PERCOM)*, Dallas-Fort Worth, Texas, Mar. 2003.

[96] A. C. Viana, A. Ziviani, and R. Friedman. Decoupling data dissemination from mobile sink's trajectory in wireless sensor networks. *IEEE Communications Letters*, Mar. 2009.

[97] Z. Vincze, D. Vass, R. Vida, A. Vidacs, and A. Telcs. Adaptive sink mobility in event-driven multi-hop wireless sensor networks. In *Proceedings of 1st International Conference on Integrated Internet Ad Hoc and Sensor Networks*, pages 30–31, May 2006.

[98] S. Wasserman and K. Faust. *Social network analysis: Methods and applications*. Cambridge Univ Pr, 1994.

[99] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2002.

[100] B. Wu, V. Goel, and B. D. Davison. Topical trustrank: using topicality to combat web spam. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 63–72, New York, NY, USA, 2006. ACM.

[101] H. Yang, F. Ye, and B. Sikdar. SIMPLE: Using swarm intelligence methodology to design data acquisition protocol in sensor networks with mobile sinks. In *Proceedings of the IEEE INFOCOM 2006*, Apr. 2006.

[102] S. Yue, Y. Xiao, X. Zhang, J. Chen, J. Zhang, and Y. Sun. a Survey of Fault tolerance in ad-hoc networks and Sensor networks. *Underwater Acoustic Sensor Networks*, page 109, 2010.

[103] Y. J. Zhao, R. Govindan, and D. Estrin. Residual energy scans for monitoring wireless sensor networks. In *In Proceedings of the IEEE Wireless Communications and Networking Conference*, pages 17–21, 2001.