MULUS : un réseau social distribué pour une consommation sélective de contenus composites

Amina Chaabane

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *L'Institut National des Sciences Appliquées/(INSA) de Toulouse*

Présentée et soutenue le *04/07/2016* **par :**

AMINA CHAÂBANE

MULUS : un réseau social distribué pour une consommation sélective de contenus composites

JURY

MYRIAM LAMOLLE	Professeur d'Université	Rapporteur
YUDITH CARDINALE	Professeur d'Université	Rapporteur
THIERRY VILLEMUR	Professeur d'Université	Examinateur
MATHIEU GINESTE	Ingénieur de Recherche	Examinateur
WASSEF LOUATI	Maître Assistant	Examinateur
ERNESTO EXPOSITO	Maître de Conférences	Co-directeur de thèse
KHALIL DRIRA	Directeur de Recherche	Co-directeur de thèse
MOHAMED JMAIEL	Professeur d'Université	Co-directeur de thèse

Écoles doctorales :

MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

EDST : Ecole Doctorale Sciences et Technologies

Laboratoires de Recherche :

LAAS-CNRS - (UPR 8001)

ReDCAD-ENIS - (LR13ES26)

Directeurs de Thèse :

Ernesto Exposito, *Khalil Drira* et *Mohamed Jmaiel*

Rapporteurs :

Myriam Lamolle et *Yudith Cardinale*

À tous ceux qui comptent pour moi...

Remerciements

C'est avec un grand plaisir que je réserve cette page en signe de gratitude et de reconnaissance à tous ceux qui m'ont assistée dans ce travail.

Je voudrais tout d'abord remercier mes directeurs de thèse M. *Mohamed Jmaiel*, Professeur à l'ENIS de Sfax, M. *Khalil Drira*, Directeur de Recherche CNRS au LAAS de Toulouse, et M. *Ernesto Exposito*, Maître de Conférences à l'INSA de Toulouse, qui n'ont pas épargné le moindre effort dans leurs encadrements de cette thèse. Également, j'exprime mes remerciements à M. *Wassef Louati*, Maître Assistant à la FSEG de Sfax. C'est grâce à leur aide précieuse et leurs conseils que j'ai pu réaliser ce travail.

Je tiens à témoigner ma gratitude et mes vifs remerciements à Mme. *Yudith Cardinale*, Professeur à l'Universidad Simón Bolívar, et Mme. *Myriam Lamolle*, Professeur à l'IUT de Montreuil - Université Paris 8, d'avoir accepté d'être les rapporteurs de ce travail.

J'exprime également ma profonde gratitude à M. *Thierry Villemur*, Professeur à l'IUT de Blagnac et à M. *Mathieu Gineste*, Ingénieur de Recherche à Thales Alenia Space de Toulouse, qui ont accepté d'être membres de mon jury de thèse.

Effectuer cette thèse au sein des deux équipes de recherche SARA au LAAS-CNRS et ReDCAD à l'ENIS m'a permis d'établir de belles relations amicales. Le temps que j'ai passé avec les membres de ces deux équipes et les échanges d'expériences avec eux ont enrichi mes connaissances. Je ne peux pas laisser passer cette occasion sans exprimer ma profonde reconnaissance à mes amis tous pour avoir contribué à guider mes pas et enrichir mon savoir.

Résumé

Avec l'évolution des nouvelles technologies, les réseaux sociaux (RS) sont fréquemment utilisés pour partager des données entre plusieurs utilisateurs. Ces RS présentent deux limitations majeures. D'une part, une grande partie des contenus partagés n'intéresse pas ces utilisateurs, ce qui augmente inutilement la consommation de la bande passante et dégrade la qualité de service de ces RS. D'autre part, ces RS utilisent une infrastructure centralisée gérée par un fournisseur de services. Ce dernier impose des contraintes liées à l'espace de stockage et aux services offerts comme il peut exiger des frais pour l'utilisation du RS.
Dans ce contexte, nous avons développé un RS distribué (RSD) en pair-à-pair (DHT) fondé sur un système publier/souscrire. Son service d'évènements est déployé sur les équipements des utilisateurs en les communicant en P2P pour surmonter les contraintes des fournisseurs de services. Notre RSD assure un partage de données sélectif où les utilisateurs reçoivent des contenus selon leurs intérêts exprimés grâce au paradigme publier/souscrire. Notre RSD permet de décrire les intérêts des utilisateurs par des souscriptions sémantiques et composites.

Pour le traitement de la sémantique, nous avons utilisé une ontologie de domaine structurée et partagée entre les utilisateurs. Nous avons proposé ensuite une méthode d'indexation de cette ontologie fondée sur les nombres premiers. Cette méthode offre un routage sémantique sur une architecture P2P structurée (DHT) de notre plateforme. Pour exprimer les intérêts composés, nous avons défini une structure de cube pour l'indexation des souscriptions composites sur les nœuds DHT. Cette structure assure le filtrage des évènements composites par une simple recherche binaire dans ce cube.

Bien que l'architecture de notre RSD assure l'auto-organisation des nœuds, la disponibilité des données et leur livraison sans perte reste encore un défi pour des nœuds très dynamiques. Nous avons alors proposé une stratégie de réplication selon la disponibilité des nœuds en cherchant à atteindre la disponibilité souhaitée par l'utilisateur.

Des expérimentations menées à l'aide du simulateur FreePastry ont montré l'efficacité de notre RSD par rapport aux RS existants, en termes de scalabilité, temps de réponse et trafic sur le réseau.

Mots-clés : réseau social, publier/souscrire, pair-à-pair, sémantique, évènement composite, forte dynamicité.

Abstract

With the evolution of new technologies, social networks (SN) are extensively used for data sharing between users. They have two major shortcomings. First, much of the shared content does not interest those users, which unnecessary increases the bandwidth consumption and degrades the SN quality of service. Second, these SNs use a centralized infrastructure managed by a service provider. This latter imposes some constraints related to storage capacity and offered services and can also involve fees for using the SN.
In this context, we developed a distributed SN (DSN) based on peer-to-peer (DHT) publish/subscribe system. It deploys the event service on user devices connected in P2P manner. Our DSN ensures selective data sharing where users receive content generated according to their interests, which are specified using the publish/subscribe paradigm. Our DSN expresses user interests with semantic and composites subscriptions.

To address the semantic aspect of subscriptions, we proposed to use a structured domain ontology, which is shared between all users. Then, we proposed an indexing method of this ontology based on prime numbers. This method provides a semantic routing on structured P2P topology (DHT) of our system. To preserve the expressivity of composite interest, we have defined a cube structure to index subscriptions on the DHT nodes. This structure provides filtering of composite events by a simple binary research performed on this cube.

Although DHT used in our DSN provides self-organizing nodes, handling data loses still remains a challenge with highly dynamic nodes. Then, we proposed a replication strategy based on nodes availability in order to maximize the availability rate desired by the user.

Experimental results using the FreePastry simulator demonstrate the outperformance of our DSN over existing SNs, in terms of scalability, latency and network overhead.

Keywords : social network, publish/subscribe, peer-to-peer, semantic, composite event, churn.

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes

Introduction Générale

Contexte et motivation

Avec l'évolution des nouvelles technologies, les réseaux sociaux (RS) sont en plein essor et jouent un rôle important dans notre vie culturelle, sociale, économique et professionnelle. Ils sont très utilisés pour partager des données (vidéo, audio, texte, etc.) entre des utilisateurs distants à travers un réseau étendu. Ceci augmente de plus en plus le contenu généré par les utilisateurs (User-Generated Content ou UGC). Ces RS sont aussi très variés par leurs caractéristiques et leurs domaines d'application. Ainsi, les besoins d'un seul utilisateur peuvent exiger l'utilisation d'un ou plusieurs RS. Ces derniers sont de plus en plus développés pour couvrir les besoins des différents utilisateurs. Cependant, ils présentent encore la limite d'expression des intérêts des utilisateurs devant la grande quantité d'UGC produite sur Internet [17]. Ces réseaux permettent généralement une consommation passive des données partagées sans que les utilisateurs expriment leurs intérêts auparavant. Ceci engendre un échange de données non intéressantes, ou un échange de données utiles se produisant à des moments inadéquats pour l'utilisateur. Quelques RS tels que YouTube offrent la possibilité d'exprimer les besoins de l'utilisateur par une simple requête. Cette dernière ne permet pas encore d'exprimer un besoin composite et ne notifie pas l'utilisateur par la présence d'un nouveau contenu cherché auparavant. Un contenu inutile devient souvent fastidieux pour les utilisateurs et le partage d'un tel contenu consomme de la bande passante et génère du trafic sur le réseau. D'où le besoin de bien exprimer les intérêts des utilisateurs du point de vue contenu et ordre de réception par une requête persistante.

De plus, les RS déployés sur Internet présentent une infrastructure peu évolutive par rapport à l'augmentation du nombre d'utilisateurs et de la quantité d'UGC. Ces RS utilisent des serveurs et des réseaux de diffusion de contenus (CDN) sous le contrôle d'une entité appelée

fournisseur de services. Ce dernier impose certaines contraintes telles qu'un espace de stockage limité, des frais pour l'utilisation du RS ou pour profiter de quelques services du RS, etc. Pour outrepasser ces contraintes, nous proposons une plateforme où les équipements des utilisateurs sont eux-mêmes les serveurs organisés en pair-à-pair. Notre plateforme permet de partager aisément les données directement entre les utilisateurs tout en respectant leurs intérêts. Ainsi, elle cherche à socialiser les abonnés autour de leurs intérêts au lieu d'être liés qu'aux abonnés qui les connaissent. Ainsi, nous aurons une augmentation proportionnelle entre le nombre d'utilisateurs avec leurs équipements et la quantité d'UGC partagé sur le RS.

Problématique

Dans ce travail de thèse, nous abordons la problématique de partage de données composites dans les RS entre des utilisateurs selon leurs intérêts. Pour mieux répondre aux besoins des utilisateurs et pour la performance de notre RS, nous prenons en compte plusieurs contraintes que nous avons regroupées en quatre thématiques complémentaires.

1. Infrastructure de RS peu évolutive

D'après notre étude bibliographique, nous avons constaté que la plupart des RS populaires utilisent une infrastructure formée par un nombre limité de serveurs sous le contrôle d'un fournisseur de services [1]. Ce dernier exige soit des frais pour l'utilisation du RS, soit des contraintes telles que la limitation de l'espace de stockage, les frais de souscription, les problèmes de sécurité et de confidentialité des données personnelles des utilisateurs, etc.
Selon les statistiques de Cisco VNI [18], plus d'un demi-milliard (563 millions) d'appareils mobiles ont été ajoutés en 2015 dont la majorité étaient des smartphones. Les appareils mobiles ont atteint 7.9 milliards en 2015 contre 7.3 milliards en 2014. Le trafic mondial de données mobiles augmentera près de huit fois entre 2015 et 2020. Le trafic de données mobiles s'élèvera à un taux de croissance annuel composé (TCAC) de 53% de 2015 à 2020, pour atteindre 30.6 exaoctets (milliards de giga-octets) par mois d'ici 2020. Par conséquent, l'infrastructure des RS existants demeure peu évolutive par rapport à l'UGC produit et l'augmentation du nombre d'utilisateurs et de dispositifs connectés [1]. Ainsi, plusieurs travaux de recherche ont dévié récemment vers la décentralisation des RS pour améliorer la qualité de leurs services [2, 3, 4, 5, 6, 7, 8, 9].

La première question est donc : comment mettre en place un RSD performant qui supporte l'augmentation du trafic de données et du nombre de dispositifs connectés ?

2. Consommation passive des contenus

La deuxième question à traiter concerne l'expression des intérêts des utilisateurs. En effet, la majorité des RS existants ne permet qu'une consommation passive des contenus partagés, c'est-à-dire ne permet pas aux utilisateurs d'exprimer leurs besoins [6, 7, 8, 9]. Avec la diversité et l'hétérogénéité des contenus partagés sur les RS, les utilisateurs reçoivent de plus en plus de données inutiles. Quotidiennement, nous remarquons qu'après réception des contenus, l'utilisateur peut se rendre compte que le contenu reçu ne convient pas à son besoin. Ceci devient ennuyeux pour les utilisateurs et résulte en une dégradation de la qualité de service sur le réseau. Quelques RS, tels que YouTube, tiennent en compte les intérêts des utilisateurs décrits par des requêtes instantanées. Ces RS ne permettent ni la description détaillée des contenus souhaités ni la notification en temps réel de nouveaux contenus disponibles.

La deuxième question est alors : comment tenir compte des intérêts des utilisateurs et leurs notifier uniquement par les contenus souhaités dès qu'ils sont disponibles sur le réseau ?

3. Manque d'expressivité pour les intérêts composites

Tenir compte uniquement des contenus échangés reste insuffisant si les utilisateurs s'intéressent à une composition particulière de contenus spécifiques. En effet, nous remarquons souvent qu'après réception des contenus sur les RS existants, l'utilisateur peut se rendre compte que le contenu reçu est souhaité à un instant bien déterminé ou avec un autre contenu. Par exemple, un utilisateur souhaite regarder un film avec une connexion gratuite ou il souhaite suivre une suite de trois leçons avec une vidéo explicative. Autrement dit, un contenu n'a de valeur pour l'utilisateur que s'il est suivi, précédé ou accompagné par un ou plusieurs autres contenus ou qu'il est reçu à un instant bien déterminé. Par conséquent, nous constatons qu'il y a fréquemment des échanges inutiles qui se produisent sans satisfaire les intérêts des utilisateurs. Il en résulte aussi une dégradation de la qualité de service (QdS) sur le réseau par les échanges inutiles. Il nous faut alors une composition des contenus selon les intérêts composites souhaités par les utilisateurs.

La troisième question est donc : comment assurer un échange de contenus sur un RSD en tenant compte des intérêts composites des utilisateurs ?

4. Livraison non efficace des contenus sur un RSD

De nos jours, l'utilisation des nouvelles technologies favorise la mobilité des utilisateurs. Ceci engendre un dynamisme fort des utilisateurs vu les connexions et déconnexions fréquentes avec des durées de sessions variées. Ainsi, nous devons bien considérer le dynamisme fort des utilisateurs lors du partage des données vu que le partage se fait directement entre eux sans passer par un serveur intermédiaire externe. Nous devons alors traiter le dynamisme fort des utilisateurs pour assurer la disponibilité des données partagées et leur livraison sans perte. Ceci présente un défi majeur pour les RSD.

La quatrième question est donc : comment assurer la disponibilité de données et une livraison sans perte sur un RSD ?

Contributions de la thèse

Dans le cadre de cette thèse, nous développons un RSD pour une consommation sélective de contenus composites. Pour atteindre cet objectif et répondre aux questions mentionnées, nos contributions reposent sur quatre axes. Dans les sections qui suivent, nous décrivons brièvement chacun d'eux.

1. Développement d'un RSD

Pour dépasser les limites et contraintes exigées par les fournisseurs de services des RS, nous proposons de développer un RS totalement distribué en pair-à-pair. La caractéristique la plus marquante de cette architecture est que la communication entre les utilisateurs s'effectue directement sans faire intervenir des serveurs centraux en utilisant les dispositifs des utilisateurs comme serveurs/gateway. Ainsi, nous aurons une augmentation proportionnelle entre le nombre d'utilisateurs connectés, les contenus partagés et le nombre de serveurs utilisés. Cette architecture permet de développer des applications distribuées efficaces et scalables, en particulier des RS qui soutiennent des millions d'utilisateurs en ligne. Nous utilisons la table de hachage distribuée Pastry [19] comme réseau P2P structuré permettant de communiquer les utilisateurs distants en assurant l'auto-organisation des nœuds qui composent le réseau et l'équilibrage de charge (le *load-balancing*) entre eux.
Notre RSD permet alors de partager aisément les données directement entre les utilisateurs.

2. Expression des intérêts

Comme les systèmes publier/souscrire ont conforté l'intérêt des utilisateurs, il est donc

intéressant de les utiliser pour résoudre le problème de fond de cette thèse (partage sélectif de données). Nous avons développé alors notre nouvelle plateforme que nous avons appelée MULUS, qui étend le système publier/souscrire Scribe [20] fondé sur la DHT Pastry [19] pour tirer profit des avantages des systèmes publier/souscrire et des réseaux P2P structurés (DHT). Notre plateforme communique les clusters des différents utilisateurs à travers une entité *Home Box Entity* (HBE). Cette entité logicielle est déployée sur les dispositifs sélectionnés de chaque cluster pour jouer le rôle de gateway/pair dans notre RS.

Avec un réseau P2P structuré, nous nous sommes retrouvés face au problème de la sémantique des requêtes envoyées sur DHT. En effet, le routage sur ce type de réseau ne permet que de vérifier l'égalité entre les clés de routage, tandis que la description sémantique permet l'utilisation des mots syntaxiquement différents mais sémantiquement équivalents. Pour résoudre ce problème, nous avons utilisé une ontologie de domaine structurée et partagée entre tous les utilisateurs. Nous avons proposé une méthode d'indexation de cette ontologie fondée sur les nombres premiers. Cette méthode offre un routage sémantique sur DHT. Le HBE est responsable de la gestion et le routage des requêtes sémantiques entre les différents utilisateurs.

3. Traitement des intérêts composites

Pour éviter les échanges qui se produisent à des moments inadéquats et pour répondre aux intérêts composites des utilisateurs, nous avons proposé de traiter la composition d'évènements sur notre système publier/souscrire en P2P. Nous avons défini une nouvelle méthode d'indexation des événements fondée sur un cube. La structure du cube assure le filtrage des évènements composites par une simple recherche binaire effectuée sur ce cube. Notre plateforme permet ainsi de décrire les intérêts composites des utilisateurs par des requêtes sémantiques (par ontologie), et persistantes (publier/souscrire), sur le réseau. Ceci permet de réduire le trafic sur le réseau en éliminant les échanges inutiles entre les utilisateurs.

4. Traitement de la disponibilité des données

La dernière contribution dans cette thèse consiste à assurer la disponibilité des données partagées et leur livraison sans perte. En fait, le problème de disponibilité de données sur les RSD est un sujet d'actualité dont peu de solutions ont été proposées. En effet, la majorité des approches proposées s'intéresse plus à la confidentialité des données qu'à la disponibilité et à la livraison sans perte. En outre, quelques approches proposent que les utilisateurs des RSD aient la liberté de stocker leurs contenus sur des équipements de leur choix ou

fassent des répliques chez des amis de leur choix. Ceci peut résoudre relativement le problème de disponibilité de données mais il reste insuffisant pour assurer une livraison sans perte. Bien que la DHT utilisée dans notre RS assure l'auto-organisation des nœuds, la livraison de données sans perte reste encore un défi. Nous avons alors proposé deux stratégies de réplication pour assurer la disponibilité de données ainsi que la livraison sans perte. La première réplique la souscription à partir d'un autre point d'accès le plus loin du point d'accès courant. Ceci permet de créer un chemin pour la souscription complètement différent du premier, ce qui permet de retrouver un chemin pour l'acheminement des publications. La deuxième stratégie fait la réplication des souscriptions sur d'autres nœuds selon leurs disponibilités en cherchant à atteindre une disponibilité totale souhaitée par l'utilisateur.

Organisation du manuscrit

Le premier chapitre constitue une étude sur les RS du point de vue architecture et caractéristiques. Nous commençons par les RS en service pour identifier les avantages et les inconvénients de ces réseaux. Puis, nous étudions les nouvelles approches proposées pour la décentralisation des RS et l'amélioration de la qualité de service des RS. Enfin, nous montrons les limites de ces réseaux et leurs inadéquations aux besoins identifiés.

Le deuxième chapitre est dédié à la proposition de la nouvelle architecture de notre plateforme MULUS. Pour mieux expliquer cette architecture, nous commençons par présenter les concepts des réseaux P2P ainsi que des systèmes publier/souscrire et leurs architectures. Ensuite, nous présentons l'architecture de MULUS. Enfin, nous décomposons nos contributions pour pouvoir les traiter dans les chapitres suivants.

Dans le troisième chapitre, nous nous concentrons sur notre première contribution qui est le traitement de la sémantique dans notre plateforme MULUS pour répondre aux besoins des utilisateurs. Nous commençons par une étude bibliographique pour étudier la sémantique dans les systèmes publier/souscrire. Ensuite, nous détaillons notre approche pour le traitement de la sémantique sur MULUS. Nous définissons la taxonomie de l'ontologie de domaine utilisée ainsi que la méthode d'indexation proposée pour le routage sémantique sur DHT. Nous détaillons l'architecture de HBE définie pour gérer les souscriptions et publications au niveau de chaque cluster d'un utilisateur. Nous expliquons aussi le protocole de communication entre les HBE des différents clusters en exploitant l'indexation de l'ontologie de domaine

proposée. Enfin, pour valider notre approche, nous présentons les résultats des simulations que nous avons menées afin de montrer que l'intégration d'un système publier/souscrire sémantique réduit considérablement le trafic sur le RSD.

Le quatrième chapitre présente la deuxième contribution de cette thèse. Dans le but de mieux répondre aux intérêts des utilisateurs, nous étudions la composition d'évènements dans les systèmes publier/souscrire P2P. Ensuite, nous proposons une entité logicielle que nous appellerons CECube intégrée dans chaque pair responsable de la publication des évènements composites. Nous détaillons les algorithmes de routage des phases de souscription et de publication utilisant CECube. Enfin, nous donnons les résultats des simulations que nous avons conduites pour valider l'apport de notre approche.

La dernière contribution de cette thèse est présentée dans le cinquième chapitre. Nous débutons par l'état de l'art sur le traitement de la disponibilité de données dans les RS ainsi que dans les systèmes d'architecture P2P structurée et nous soulignons alors les points faibles des approches trouvées dans la littérature. Nous proposons par la suite deux stratégies de réplications pour assurer la disponibilité des données partagées et leur livraison sans perte sur MULUS. Nous expliquons la première stratégie (*Complementary Digit*) qui consiste à répliquer la souscription à partir d'un autre point d'accès le plus loin du point d'accès de l'utilisateur et nous donnons une description détaillée de cette stratégie. Ensuite, nous détaillons la deuxième stratégie fondée sur la disponibilité des nœuds DHT afin d'atteindre une disponibilité souhaitée par l'utilisateur. Enfin, nous comparons ces deux stratégies pour prouver que la stratégie de réplication selon la disponibilité de nœuds est la plus efficace pour minimiser le nombre de répliques, augmenter le taux de disponibilité et améliorer la qualité du service au moment de la livraison.

Le dernier chapitre présente la conclusion générale de cette thèse ainsi que les perspectives futures de nos recherches à ce sujet.

Chapitre

1 État de l'art : Réseaux Sociaux (RS)

1.1 Introduction

Avec les innovations technologiques, les réseaux sociaux (RS) émergent dans notre vie d'une manière croissante au point de réduire l'utilisation des courriers électroniques parfois même dans la vie professionnelle. Le partage de données, à travers ces RS, devient de plus en plus gourmand en termes de trafic sur Internet selon les statistiques de Cisco VNI [18]. Les internautes ne cessent plus de partager, copier, créer et procéder à des mixages et manipulations de contenus. En raison de l'émergence de divers appareils portables, le nombre d'utilisateurs sur les différents RS augmente exponentiellement. La tendance d'adoption de RS crée divers défis de scalabilité et de gestion pour les fournisseurs de RS tels que la disponibilité de service, la tolérance aux pannes et la sécurité des données ; d'où un besoin fort de solutions efficaces pour pouvoir gérer la tendance croissante des RS.

Dans ce chapitre, nous présentons brièvement la définition d'un RS, ses principaux services, l'historique des RS ainsi que leurs architectures et leurs caractéristiques. Nous étudions aussi les RS populaires tels que Facebook, LinkedIn et Twitter pour faire ressortir leurs limites. Puis, nous recensons les solutions existantes pour la décentralisation des RS. Avant de conclure, nous montrons les limites de ces solutions par rapport aux exigences des utilisateurs.

1.2 Définition d'un RS

"Un RS est un ensemble de relations entre un ensemble d'acteurs", selon Michel Forsé [21]. Ces acteurs sont des entités sociales (individu, groupe ou organisation), reliées par des relations sociales. Ces relations peuvent être de nature parentale, amicale et sentimentale ou

d'autres natures (relation de travail, d'affaires, une simple connaissance, etc.). Elles peuvent se former à travers des contacts directs ou d'intermédiaires. Sur Internet, un RS interconnecte les acteurs d'un RS via Internet selon des relations sociales entre eux.

1.3 Services offerts par les RS

En général, un RS est un site Web qui :

- agit comme un concentrateur pour les individus pour établir des relations avec d'autres personnes (amis, collègues, etc.). Chaque utilisateur construit un groupe d'amis avec lesquels une connexion est partagée.
- comprend une large gamme d'outils pour les utilisateurs pour construire un sens de la communauté de manière informelle et volontaire. Les utilisateurs en ligne interagissent les uns avec les autres, contribuent avec des informations dans l'espace d'information commun et participent à différentes activités interactives entre eux telles que le partage de photos, le *tagging*, etc.
- contient des composants spécifiques permettant aux utilisateurs de : définir leurs profils en ligne, grouper leurs connaissances telles que les amis, les collègues, etc., recevoir des notifications sur les nouvelles activités, participer aux activités interactives et contrôler les paramètres d'accès à leurs profils.

1.4 Historique des RS

Avec l'émergence du Web 2.0, les utilisateurs finaux sont placés au cœur de diverses technologies Web, qui ont tendance à remodeler l'avenir du WWW en termes de structure, conception et utilité. Dans ce contexte, les RS en ligne apparaissent comme un nouveau type d'application sur Internet, qui peut être considéré comme une extension naturelle des applications Web établissant et gérant des relations explicites entre les utilisateurs pour la majorité des RS. Plus précisément, un RS se compose d'utilisateurs qui communiquent les uns avec les autres en ligne par diverses manières. Les RS ont eu beaucoup de succès ces dernières années et le nombre d'utilisateurs connectés sur ces réseaux augmente d'une façon exponentielle. Aujourd'hui, nous témoignons la croissance rapide d'une grande variété de RS, qui publient les contenus générés ou modifiés par les utilisateurs et qui permettent

aux utilisateurs d'annoter le contenu publié par des étiquettes, des commentaires et des recommandations [1].

Le premier RS le plus connu, nommé *SixDegrees.com*, est lancé en 1997. L'origine de son nom provient de la théorie de *six degrés de séparation*. Cette théorie évoque la possibilité que toute personne de l'univers peut être reliée à n'importe quelle autre personne, à travers une chaîne du réseau relationnel entre elles comprenant au plus cinq autres maillons intermédiaires. À travers SixDegrees.com, les utilisateurs peuvent créer leurs profils, avoir un groupe d'amis et fournir des informations à leur communauté. Bien que ce réseau ait attiré des millions d'utilisateurs, il n'a pas pu évoluer et a été stoppé en 2000. Le fondateur de SixDegrees.com estime qu'il était en avance sur son temps. Depuis 2003, nous avons assisté à une révolution et une assimilation de sites de RS qui ont établi la plupart des sites les plus populaires de nos jours. Cette révolution a apporté un changement radical sur l'activité des utilisateurs, leurs cultures et le domaine de la recherche sur le WWW [22]. La Figure 1.1 présente l'évolution des RS au cours de la dernière décennie.

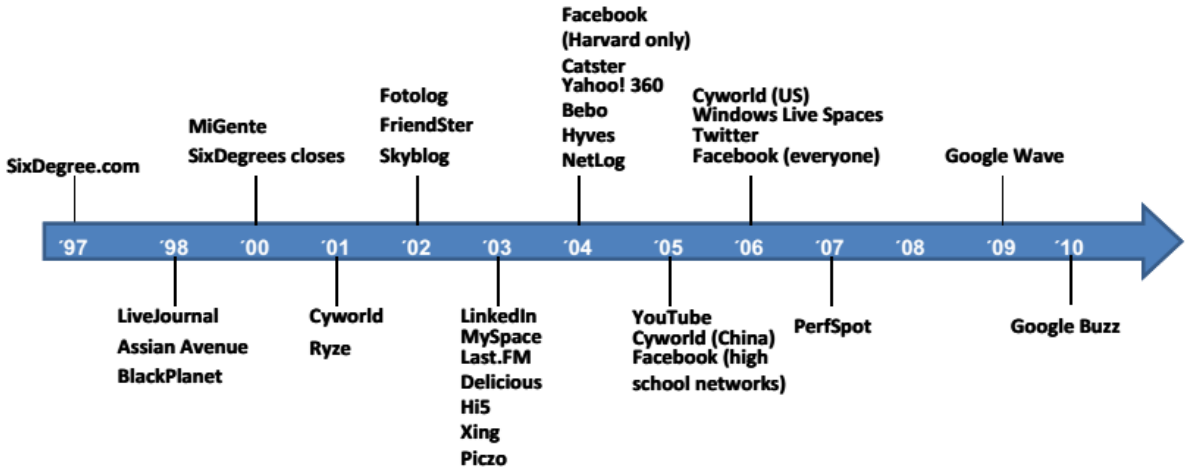


Figure 1.1 – Chronologie des RS [1]

Selon les dernières recherches de Nielson [1] faites en 2010, les sites des RS et des blogs sont classés comme la quatrième activité la plus populaire sur Internet. En effet, plus des deux tiers de la population mondiale connectée participent aux RS et aux blogs. En outre, les médias sociaux ont pris de l'avance sur l'e-mail dans le classement des activités les plus

1. Nielsen Company :
http ://blog.nielsen.com/nielsenwire/wpcontent/uploads/2009/03/nielsen_globalfaces_mar09.pdf

populaires en ligne. Une autre constatation intéressante est que les temps de session sur les RS et les blogs atteint 10% de tout le temps passé en navigation. Ces statistiques suggèrent que les RS sont devenus un élément fondamental sur le Web à travers le monde.

Le progrès important apporté aux sites de RS comme Facebook, Myspace, Flickr, LinkedIn et YouTube, et la principale force motrice de leur succès, sont expliqués par le fait que les sites de RS mettent l'accent sur les intérêts des utilisateurs pour développer leurs services. Par conséquent, l'objectif principal des RS est de fournir des fonctionnalités de réseautage social comme un service de base pour une variété d'applications et de services de haut niveau. En outre, les RS en ligne sont à l'issu de nouveaux problèmes intéressants et des défis pour la recherche dans un environnement qui devient de plus en plus complexe [23, 24]. Aujourd'hui, les RS sont devenus l'objet de nombreuses entreprises en démarrage, offrant aux utilisateurs la possibilité de créer, rechercher et gérer leurs propres communautés de RS.

1.5 Architecture de base des RS

Une architecture de référence pour les RS déployés sur Internet est présentée dans la Figure 1.2. Le système entier est formé par les couches suivantes situées au dessus de la couche hardware et la couche du système d'exploitation :

- ***Data Storage Layer :*** cette couche est formée par deux composants principaux : le *Storage Manager* qui est chargé de stocker efficacement les informations des graphes sociaux en assurant la manutention de charges croissantes de bases de données. Ceci est habituellement réalisé en adoptant une mémoire cache distribuée. Le deuxième composant est appelé Data Store. Il comprend les éléments de stockage qui mémorisent des informations sur le service du RS. Ces Data Stores peuvent être des bases de données Multimédia, des bases de données des Profils utilisateurs, etc.
- ***Content Management Layer :*** cette couche est responsable de trois tâches principales. Tout d'abord, il s'agit de collecter des informations sociales provenant des autres RS pour les incorporer dans un *Content Aggregator* qui collecte et organise les contenus reçus des médias sociaux et les distribue aussi vers les autres RS. Deuxièmement, il facilite la maintenance et la récupération du graphe présentant le contenu social à travers le *Data Manager*. La troisième tâche *Access Control Manager* consiste à contrôler

l'accès des utilisateurs par la création et la maintenance du schéma de contrôle d'accès.

– ***Application Layer :*** chaque site Web de RS supporte plusieurs services implantés par la couche application, comme la recherche, le flux d'actualités, les accès mobiles, etc. Les services de cette couche communiquent avec le *Data Manager* et l'*Access Control Manager* dans le but d'analyser et de gérer le graphe du contenu social. Les applications sont fournies aux utilisateurs à travers l'*Application Manager*. L'*Application Manager* facilite l'interaction de l'utilisateur à travers un ensemble d'API. Ce composant contient aussi un *Service Framework* pour le développement de services évolutifs franchissant les obstacles linguistiques.

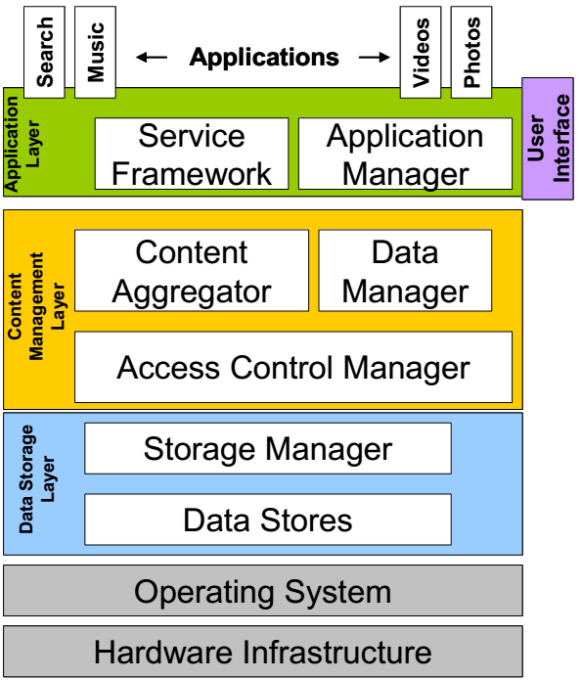


Figure 1.2 – Architecture de base d'un RS [1]

Les utilisateurs interagissent avec une plateforme de RS via des requêtes HTTP. Chaque utilisateur peut soit être inscrit comme utilisateur autorisé ou être anonyme. Les utilisateurs enregistrés ont généralement plus de droits que ceux anonymes (pour commenter des contenus publiés, télécharger des éléments multimédias, etc.).

Le module de contrôle d'accès est chargé de traiter les paramètres de confidentialité et de sécurité des utilisateurs des RS. La plupart des recherches dans le domaine de confidentialité et sécurité des RS se sont concentrées sur le développement des techniques de protection des

renseignements personnels en fournissant des solutions à des problématiques telles que la propriété des informations personnelles et la protection de la vie privée [25, 26, 27, 28, 29].

Chaque plateforme de RS se compose de plusieurs serveurs d'applications, qui offrent un ensemble de services et d'API. Un utilisateur envoie une requête à la plateforme du RS, qui la transmet au serveur d'application approprié. Un équilibreur de charge est responsable de surveiller les serveurs d'applications de RS, d'équilibrer la charge des requêtes, de protéger le système contre les pannes par relais entre ces serveurs, et de transmettre les requêtes aux serveurs d'applications [1]. Quant à la distribution de contenu, des serveurs de cache accélèrent les applications Web dynamiques en allégeant la charge des serveurs d'applications.
De plus, les RS présentent des exigences de Qualité de Service (QdS) importantes par rapport aux applications Web traditionnelles. Parmi les principales différences entre les applications Web et les plates-formes de RS, nous citons les relations et la confiance entre les utilisateurs, qui changent au fil du temps. Ce comportement dicte la conception d'algorithmes de placement des données, de réplication et de distribution dans les RS. Une autre différence importante est que les RS impliquent un grand nombre de fichiers de petites tailles qui doivent être fréquemment consultés et mis à jour par un grand nombre d'utilisateurs et impliquent la propagation des mises à jour de ces fichiers pour garantir la cohérence des données. Pour améliorer davantage les performances du système, les plates-formes de RS distribuent leurs contenus sur les réseaux de distribution de contenu (*Content Distribution Networks*) comme *Akamai* [30], ou sur des infrastructures de Cloud. Avec un réseau de diffusion de contenu (*Content Delivery Network (CDN)*) [31], les requêtes des utilisateurs sont automatiquement acheminées aux serveurs périphériques, où les contenus des serveurs d'origine sont répliqués. Généralement, ces serveurs sont connectés à Internet à travers des dorsales Internet.
Enfin, les Data Stores des RS peuvent être centralisés ou distribués au travers de multiples domaines administratifs. Les RS centralisés soulèvent des inquiétudes concernant la protection des informations personnelles et la scalabilité.

1.6 Classification des RS

Une étude exhaustive des aspects des RS existants nous permet de déterminer la classification présentée dans la Figure 1.3 [1]. Nous retrouvons quatre grandes branches pour décrire un tel réseau social. La première branche couvre le champ d'application des RS en termes

d'activités (*Scope*). La deuxième branche concerne le modèle de données des RS, puisque le modèle de données représente la façon dont les données sont stockées dans un RS. La troisième branche (*Plateforme*) catégorise les RS selon l'hébergement et la distribution de contenus des serveurs d'applications. La quatrième branche caractérise le réseau relationnel formé entre les utilisateurs des RS.

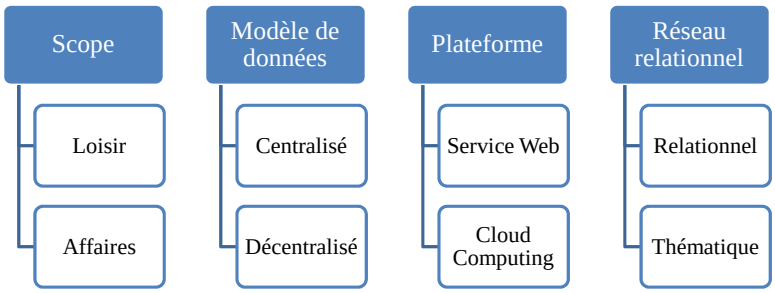


Figure 1.3 – Classification des RS

Du point de vue champ d'application, les RS peuvent être classifiés en deux catégories :

- ***RS de "loisirs" :*** la plupart des RS sont des RS de loisirs. Leur rôle est le partage des loisirs qui peuvent être des jeux, des informations, des photos, des vidéos ou des savoirs dans tous les domaines de la vie courante entre des communautés d'amis. Les plus populaires de cette catégorie sont Facebook, Myspace, Hi5 et Flickr.
- ***RS d' "Affaires" :*** leur rôle est de relier les professionnels du monde entier pour les rendre plus interactifs et productifs. Grâce au RS d'affaires, les utilisateurs enregistrés créent des profils qui résument leurs expertises et leurs activités professionnelles. Les RS indicatifs dans cette catégorie sont LinkedIn et Xing.

La deuxième dimension présentée par cette taxonomie est le modèle de données (Data Model) suivi par les fournisseurs de RS. Ici, nous identifions les catégories suivantes :

- ***Data Model centralisé :*** les données des RS centralisés sont stockées entièrement sur une proximité physique qui peut être un Cluster ou un Data Center. Ceci concentre les données des utilisateurs sous un seul domaine administratif. La plupart des RS s'appuient sur le stockage et la fonctionnalité centralisés vu que leur souci est de soulever des préoccupations concernant la protection de la vie privée des utilisateurs et leur capacité de supporter une base croissante d'utilisateurs et d'applications.
- ***Data Model décentralisé :*** les données des RS décentralisés sont distribuées à tra-

vers multiples domaines administratifs [2, 32]. Les serveurs d'applications s'exécutent sur les machines de bureau (les pairs) appartenant aux utilisateurs. En général, l'hébergement de données personnelles sur les pairs est plus sécurisé que de déléguer leur contrôle à un fournisseur de services tiers. En outre, ce modèle est moins cher que l'acquisition d'équipements centralisés dédiés. L'inconvénient majeur de cette approche est que les pairs ne sont pas disponibles en permanence. Les pairs sont sujets à des défaillances, des redémarrages, des power-offs, des déconnexions de réseau, etc. [33]. Une discussion plus détaillée sur les RS décentralisés sera donnée dans la section 1.9.

La troisième branche de cette classification présente la plateforme d'exécution des RS. Nous identifions les deux classes suivantes :

- ***RS fondé sur une application Web :*** les serveurs d'applications hébergent les sites Web qui offrent un ensemble de services et API. Dans un tel RS, l'équilibreur de charge équilibre la charge des requêtes, gère tout relais et transmet les demandes aux serveurs d'applications appropriées. Dans les régimes fondés sur le Web, la plupart des services de RS sont gratuits pour les utilisateurs.
- ***RS fondé sur le Cloud :*** les serveurs d'applications sont hébergés par une infrastructure informatique utilitaire telle qu'Amazon Elastic Compute Cloud (EC2). Dans un tel RS, chaque utilisateur stocke ses propres données sur une instance de machine virtuelle personnelle, appelée *Virtual Server*. Les principaux avantages de ce RS sont sa haute disponibilité et la sécurité des informations puisque chaque utilisateur conserve ses données personnelles dans un serveur virtuel résidant dans un environnement de Cloud Computing. En outre, les RS de Cloud Computing sont généralement intégrés aux infrastructures CDN (par exemple, Amazon Cloud Front intègré avec Amazon EC2). Ceci résulte en une distribution de contenus aux utilisateurs finaux avec une faible latence et une vitesse de transfert très élevée. D'autre part, l'hébergement de données dans un Cloud augmente les coûts dus à l'utilisation d'une infrastructure commerciale.

Finalement, la taxonomie couvre le réseau relationnel formé par le RS qui distingue ces deux classes :

- ***RS orienté utilisateur (relationnel) :*** les RS de cette catégorie mettent en valeur les relations sociales. Ainsi, le partage de contenus se fait principalement entre les utilisateurs qui appartiennent à la même communauté. Les RS de cette catégorie sont

Facebook, MySpace et LinkedIn.

– ***RS fondé sur le contenu (thématique) :*** le réseau des utilisateurs n'est pas déterminé par les relations sociales sous-jacentes, mais par les intérêts communs entre les utilisateurs. Les RS de cette catégorie sont les blogs, les forums et les services de partage (par exemple, YouTube).

1.7 RS populaires

Cette section détaille les RS existants et les plus populaires. Pour chaque plateforme de RS, nous présentons un bref historique, son architecture et son modèle économique (*Business Model*) pour enfin tirer les points forts et faibles des différents RS déployés sur Internet.

1.7.1 Facebook

Histoire et portée : Facebook [2] a été fondé par Mark Zuckerberg quand il était étudiant en psychologie à l'Université de Harvard. En Février 2004, Zuckerberg a lancé "Facebook" ; le nom a été tiré des albums photo regroupant les photos profils de tous les élèves, prises en début de l'année universitaire. Ces albums sont appelés "trombinoscopes" en français et "Facebooks" en anglais. À sa création, il était un RS réservé aux élèves de l'Université de Harvard. En septembre 2006, Facebook a été étendu au-delà des établissements d'enseignement et s'est ouvert au monde entier. Aujourd'hui, Facebook est l'une des destinations en ligne les plus populaires avec plus de 300 millions d'abonnés, qui passent en moyenne 20 minutes par jour sur ce site et qui contribuent par 4 milliards d'éléments d'information, 850 millions de photos et 8 millions de vidéos chaque mois. Il a été dédié initialement à une fonction de loisirs plutôt qu'à un usage professionnel. Il est à la fois orienté contenu et orienté utilisateur. Les utilisateurs créent leurs réseaux par la connaissance d'un groupe d'amis et ils peuvent aussi créer des réseaux fondés sur des intérêts communs. Pour un groupe fondé sur l'intérêt commun, les autres utilisateurs doivent être inscris à ce groupe et ne peuvent profiter que des informations partagées par le propriétaire du groupe. Pour un utilisateur inscrit sur Facebook, il peut ajouter des amis et leur envoyer des messages, partager des photos, vidéos, liens, mettre à jour son profil personnel et consulter le partage de ses amis.

2. www.facebook.com

Lors du partage de données, le propriétaire peut définir leur visibilité. Un utilisateur de Facebook peut consulter les données partagées par son groupe d'amis sans savoir le contenu auparavant s'il l'intéresse ou non. Ce type de partage est aussi limité souvent aux groupes d'amis.

Conception architecturale : l'architecture du modèle de données de Facebook est centralisée et fondée sur un modèle hiérarchique d'application Web en PHP avec une couche de mise en cache de données. La couche de mise en cache est fournie via le logiciel open source Memcached. En ce qui concerne la couche d'application (Application Layer), Facebook utilise un framework RPC (*Remote Procedure Call*) open source d'Apache appelé *Thrift*, qui est utilisé pour RPC en temps réel et avec une faible latence et pour le stockage persistant de données structurées provenant d'une variété d'applications, telles que la recherche, le fil d'actualité, etc. Le langage RPC est influencé par l'IDL de CORBA. Facebook soutient divers services de back-end qui utilisent les frameworks Hadoop, Scribe et Hive. Quant au *Data Storage Layer*, les données telles que les photos, les statuts et les commentaires sont périodiquement stockées dans des dépôts centraux et des bases de données de catégorie NoSQL (Cassandra [3]). Concernant la plateforme d'exécution, les serveurs d'applications de Facebook s'exécutent sur les infrastructures fondées sur des applications Web et sur *le Cloud Computing* (Amazon EC2).

Modèle économique : puisque l'inscription à Facebook est gratuite, son modèle économique dépend de la publicité. Ce modèle économique est fondé sur le principe que les inscriptions gratuites construisent un public ayant des intérêts et des besoins distincts que les annonceurs paieront pour l'atteindre. Facebook prend en charge deux politiques de publicité : le *Pay for Clicks* permet aux annonceurs de préciser un certain montant qu'ils sont prêts à payer chaque fois qu'un utilisateur clique sur leur annonce, alors que, le *Pay for Views* permet aux annonceurs de spécifier combien ils sont prêts à payer pour 1000 vues de leur annonce. En outre, Facebook offre la possibilité de cibler les annonces à des groupes d'utilisateurs spécifiques. Cela permet aux annonceurs de rédiger des textes d'annonce plus personnalisés, rendant leurs annonces plus attrayantes pour les utilisateurs qu'ils atteignent. Facebook prend en charge plusieurs options de ciblage telles que la localisation, les anniversaires, les préférences et les intérêts. Facebook a connu une réussite très importante qui est la suivante : "Pendant 12 mois, CM Photographics a généré près de 40 milles dollars de

3. http ://sql.sh/sgbd/cassandra

revenus directement d'un investissement publicitaire de 600 dollars sur Facebook. Parmi les utilisateurs de Facebook qui ont été dirigés vers le site Web de CM Photographics suite aux annonces, 60% sont devenus des prospects qualifiés".

1.7.2 MySpace

Histoire et portée : MySpace[4] est fait essentiellement pour le loisir. Il a été développé en Août 2003 par les employés d'eUniverse qui ont cherché à imiter les caractéristiques populaires du RS Friendster. Les premiers utilisateurs de MySpace étaient des employés d'eUniverse. En Juillet 2005, MySpace a été vendu pour 580 millions dollars à News Corporation de Rupert Murdoch. MySpace est devenu le site le plus populaire de RS aux États-Unis en Juin 2006. Aujourd'hui, MySpace est l'un des RS ayant une croissance la plus rapide couvrant 300 millions d'utilisateurs. C'est un RS orienté utilisateur, où chaque utilisateur enregistré peut partager entre amis son profil personnel, ses photos, ses vidéos, ses liens, etc. L'utilisateur peut également utiliser des jeux et mettre à jour son profil personnel pour informer ses amis de ses actualités. MySpace gère également la gestion d'évènements, tels que des lancements de produits et d'albums pour les grandes étiquettes qui fournissent des flux de revenus supplémentaires.

Conception architecturale : l'architecture de MySpace est centralisée et fondée sur ASP.Net 2.0 avec une couche de mise en cache de données et d'un ensemble de composants de services. Plus précisément, MySpace est composé de plus de 500 serveurs Web exécutant Windows 2003 / IIS 6.0 / APS.NET. Les Data Stores de MySpace se composent de 1200 serveurs de cache supportant Windows 2003 64-bit avec 16 Go d'objets mis en cache dans la mémoire RAM, et 500 serveurs de base de données exécutant Windows 64 bits et SQL Server 2005. Comme Facebook, la couche de mise en cache est fournie via le logiciel open source Memcached. Pour le Data Storage Layer, les données sont stockées dans des dépôts centraux et des bases de données relationnelles. Pour la plateforme d'exécution, les serveurs d'applications de MySpace s'exécutent sur des applications Web et sur le Cloud Computing (Amazon EC2).

Modèle économique : l'abonnement à MySpace est également gratuit. Par conséquent, le modèle économique de MySpace est également tributaire de la publicité. De même que Facebook, MySpace utilise deux politiques publicitaires : *Pay for Clicks* et *Pay for Views*.

4. www.myspace.com

Pour plus de pertinence, MySpace utilise un ensemble riche de données démographiques et comportementales. Ceci permet aux annonceurs de cibler leur public en se basant sur des qualités, des intérêts et des activités. Tous les services sont offerts gratuitement aux utilisateurs. Les utilisateurs peuvent créer de nouveaux contenus pour attirer, à leur tour, de nouveaux utilisateurs. Cela a encouragé plusieurs entreprises à utiliser la plateforme MySpace pour la publicité de leurs produits.

1.7.3 Hi5

Histoire et portée : Hi5[5] a été fondé en 2003 par l'entrepreneur Ramu Yalamanchi, qui est aussi actuellement le chef de la direction de Hi5. Selon[6], Hi5 était le troisième site le plus populaire de réseautage social en termes de visiteurs mensuels en 2008. Aujourd'hui, Hi5 est disponible en 50 langues et il est l'une des destinations en ligne les plus populaires de loisir avec plus que 60 millions d'utilisateurs actifs. Hi5 fournit une plateforme robuste aux développeurs tiers pour intégrer des jeux, des contenus et d'autres applications. Hi5 est un RS en ligne orienté utilisateur (relationnel) où chaque utilisateur enregistré peut télécharger des photos, des vidéos, des chansons, des renseignements personnels et les partager avec ses amis.

Conception architecturale : l'architecture de Hi5 est fondée sur une architecture n-tiers. Le Data Store de Hi5 se compose de serveurs de bases de données PostgreSQL. Hi5 fonctionne en utilisant un logiciel open-source sur une plateforme Linux. Plus précisément, Hi5 utilise des serveurs Linux exécutant SuSE Enterprise Linux, les serveurs Web Apache et Lighttpd, le serveur proxy Squid pour économiser la bande passante Internet et accélérer le service Web, les serveurs d'application Java Resin et Tomcat, Struts pour Modèle-Vue-Contrôleur (MVC), Spring pour le framework d'application Java, iBatis pour le Mapping Object-Relational, la bibliothèque Lucene pour l'indexation et Enunciate comme framework de déploiement de services Web. La couche de mise en cache est fournie via le logiciel open source memcached. En ce qui concerne la couche de stockage de données (Data Storage Layer), les données sont stockées dans une base de données objet-relationnel centralisée du système PostgreSQL. Toutes les requêtes sont gérées de manière centralisée dans des fichiers XML. Quand à la plateforme du RS, des serveurs d'applications s'exécutent sur une

5. www.hi5.com
6. www.comscore.com

infrastructure Web.

Modèle économique : comme les autres RS, la charge de Hi5 est supportée par les biens de la publicité. Hi5 est en collaboration avec SponsorSelect - un réseau de publicité de haute gamme qui réinvente le ciblage comportemental - afin de permettre à ses utilisateurs de choisir la publicité qu'ils souhaitent voir. En permettant aux utilisateurs l'auto-sélection de la publicité, la publicité devient plus pertinente pour les internautes et plus rentable pour les annonceurs et les éditeurs (un éditeur affiche des annonces, liens texte, ou des liens de produits sur son site Web).

1.7.4 Flickr

Histoire et portée : Flickr [7] est le RS le plus populaire dédié au partage des photos et des vidéos, où des millions de photos sont téléchargées, étiquetées et organisées par plus de 8.5 millions d'utilisateurs Web enregistrés. Flickr a été fondé par Stewart Butterfield et Caterina Fake. En Février 2004, Flickr a été lancé par Ludicorp qui est une société basée à Vancouver. En Mars 2005, Flickr a été racheté par Yahoo !. Flickr est un RS orienté utilisateur et il est destiné pour le loisir. Sur Flickr, chaque utilisateur propose/sélectionne de nouvelles étiquettes (tags) pour une photo ou une vidéo particulière et le système suggère des étiquettes relatives à l'utilisateur sur la base des étiquettes que l'utilisateur ou d'autres personnes ont déjà utilisées auparavant. L'objectif principal est de proposer de nouvelles méthodes de rangement pour les photos/vidéos et de permettre aux participants de les partager avec des personnes de leurs choix.

Conception architecturale : Flickr est fondé sur une application Web avec PHP et une couche de mise en cache de données. Le magasin de données de Flickr se compose de 62 bases de données MySQL à travers 124 serveurs, avec environ 800000 comptes d'utilisateurs par paire de serveurs. Les bases de données MySQL sont hébergées sur des serveurs Linux, avec une plateforme logicielle qui inclut Apache, PHP, Shards, Memcached, Squid, Perl et Java. Les outils d'administration du RS comprennent Anglia pour la surveillance du système distribué et Cvsup pour la distribution et la mise à jour des collections de fichiers à travers le réseau. Flickr supporte des outils pour le traitement d'images (ImageMagick) et leur déploiement (SystemImager). Les données (photos, vidéos et étiquettes) sont stockées dans des dépôts centraux avec une base de données relationnelle MySQL. Pour la plateforme

7. www.flickr.com

d'exécution, les serveurs d'applications de Flickr s'exécutent sur des applications Web et sur le Cloud Computing (Amazon EC2).

Modèle économique : avec 3 milliards de photos, Flickr est le plus grand répertoire de photos numériques et de vidéos sur le Web. Avoir un compte sur Flickr est gratuit, mais Flickr facture aux utilisateurs qui veulent avoir un compte professionnel et plus sophistiqué. Flickr suit la stratégie commerciale "freemium" qui combine une offre gratuite en libre accès "free" et une offre payante de haut de gamme "premium".

1.7.5 LinkedIn

Histoire et portée : LinkedIn [8] a été fondée en 2003 par Reid Hoffman. LinkedIn est un réseau de professionnels du monde entier, représentant 150 secteurs d'activités issus de 200 pays. En Décembre 2009, il comptait plus de 55 millions d'utilisateurs enregistrés et il était disponible en quatre langues. Il est utilisé pour tout ce qui est en relation avec la vie professionnelle tel que la recherche d'emploi, le recrutement, le développement des affaires, etc. Grâce à LinkedIn, les utilisateurs peuvent trouver des emplois et des occasions d'affaires. Tandis que les employeurs peuvent afficher et publier des offres d'emploi pour des candidats potentiels. LinkedIn est un RS orienté utilisateur où les utilisateurs enregistrés créent leurs réseaux en envoyant des invitations personnelles. Une caractéristique clé de LinkedIn est que les utilisateurs enregistrés peuvent être recommandés par d'autres dans leurs réseaux de contacts.

Conception architecturale : le modèle de données de LinkedIn est centralisé. Il s'agit d'une architecture ouverte qui consiste en une application Web monolithique. Le Data Store maintient un ensemble de bases de données ainsi que le graphe du RS. LinkedIn s'exécute sur le système d'exploitation Solaris, utilise Tomcat et Jetty comme serveurs d'applications, Oracle et MySQL comme bases de données, Spring pour le Framework d'application Java, la bibliothèque Lucene pour l'indexation et ActiveMQ pour Java Message Services (JMS). Les applications Web fournissent l'interface graphique pour l'utilisateur (GUI) et mettent à jour les bases de données. Pour la plateforme d'exécution, les serveurs d'applications de LinkedIn s'exécutent sur une infrastructure Web.

Modèle économique : l'adhésion au RS LinkedIn est libre. En outre, LinkedIn propose une version premium qui fournit plus d'outils pour trouver et atteindre les "bonnes"

8. www.linkedin.com

personnes. Avec un compte premium, les utilisateurs peuvent entrer en contact avec des recruteurs, promouvoir et développer des activités professionnelles, multiplier les opportunités commerciales, trouver et recruter des talents et d'autres possibilités non accessibles aux utilisateurs ordinaires.

1.7.6 Twitter

Histoire et portée : Twitter[9], fondé par Jack Dorsey, Biz Stone et Evan Williams en Mars 2006 et lancé publiquement en Juillet 2006, est un RS et outil de Microblogage (*microblogging* en anglais) qui permet aux utilisateurs de publier leurs dernières actualités en messages courts. Un message est limité à 140 caractères (appelé *tweet*) et peut être envoyé gratuitement, sur Internet, par SMS ou par messagerie instantanée. Les tweets sont livrés aux abonnés de l'auteur qui sont connus comme leurs "suiveurs". Les expéditeurs peuvent restreindre l'envoi de leurs tweets à des suiveurs spécifiques ou, par défaut, permettre un accès ouvert.

Conception architecturale : le modèle de données de Twitter est centralisé et fondé sur le Framework Ruby on Rails comme outil pour réaliser des applications Web avec une couche de mise en cache de données. La couche de mise en cache est fournie via le logiciel open source Memcached. Pour le Data Storage Layer, les données sont stockées périodiquement aux dépôts centraux et des bases de données relationnelles (MySQL). En outre, il supporte des outils de surveillance des ressources réseau (Munin et Nagios) pour le suivi des ressources et le diagnostic des problèmes de performances sur les serveurs en temps réel. Pour la plateforme d'exécution, Twitter s'exécutent sur une infrastructure Web.

Modèle économique : Twitter est gratuit pour tous les internautes. Contrairement à la plupart des RS, Twitter ne suit aucune politique publicitaire. En outre, il n'offre pas de comptes premiums. En effet, les dépendances de Twitter sont très faibles puisqu'il n'a pas besoin d'une infrastructure complexe.

1.7.7 YouTube

Histoire et portée : YouTube[10], fondée par Steve Chen, Chad Hurley et Jawed Harim en Février 2005, est un RS qui permet aux utilisateurs de publier leurs vidéos. En Novembre

9. www.twitter.com
10. www.youtube.com

2006, YouTube a été racheté par Google pour 1.65 milliard dollars. Récemment, YouTube a été classé comme le quatrième site le plus visité sur Internet. Selon comScore, YouTube est le fournisseur de vidéos en ligne le plus dominant aux États-Unis. ComScore estime que 20 heures de nouvelles vidéos sont téléchargées sur le site chaque minute. En Mars 2008, le coût de bande passante de YouTube a été estimé à environ 1 million dollars par jour. YouTube est un RS pour le loisir. Les utilisateurs peuvent regarder toutes les vidéos, tandis qu'ils ne sont autorisés à télécharger et envoyer qu'un nombre limité de vidéos. YouTube est un RS orienté contenu puisque les utilisateurs de réseau cherchent à regarder les vidéos qui sont le résultat d'une requête bien déterminée spécifiée par eux-mêmes. Avec les dernières versions de YouTube, les utilisateurs peuvent s'abonner pour recevoir les nouvelles sur un thème particulier. Cependant, il est souvent utilisé comme un moteur de recherche sur les vidéos par les internautes.

Conception architecturale : le modèle de données de YouTube est décentralisé et fondé sur le système de stockage distribué, appelé BigTable. BigTable, développé par Google, est destiné au stockage efficace de données structurées et géantes. Pour améliorer davantage ses performances, BigTable est utilisé avec MapReduce qui est un framework pour exécuter des calculs parallèles. Concernant la livraison de contenu, les contenus les plus populaires sont déplacés vers un fournisseur CDN (Akamai), alors que, les contenus moins populaires sont délivrés par les serveurs de YouTube. Un cache Web distribué est utilisé à plusieurs niveaux, pour diminuer le temps de latence et améliorer les performances globales. Un contrôleur d'application Web NetScaler est utilisé pour l'équilibrage de charge et la mise en cache du contenu statique. Pour le Data Storage Layer, les données sont stockées périodiquement dans des bases de données MySQL. Pour la plateforme d'exécution, les serveurs d'applications de YouTube s'exécutent sur des applications Web et sur le Cloud Computing (Amazon EC2).

Modèle économique : YouTube est gratuit pour tous les utilisateurs (abonnés ou non). Le modèle économique de YouTube dépend de la publicité (c'est-à-dire, Google AdSense). Google AdSense utilise une technologie de recherche sur Internet pour diffuser des publicités en fonction du contenu du site, de la localisation géographique de l'utilisateur, des étiquettes des utilisateurs et d'autres paramètres. Ceux qui veulent faire de la publicité avec le système de publicité ciblée de Google peuvent s'inscrire via *AdWords*. *AdWords* utilise *Pay for Clicks*, et la publicité ciblée. Ainsi, le modèle économique de YouTube repose sur la collaboration de masse. Fournir un accès gratuit à ses utilisateurs résulte en un nombre croissant d'utilisateurs

et, par conséquent, en l'augmentation des bénéfices grâce à des taux de publicité et un nombre d'annonceurs accrus.

1.7.8 Synthèse

Selon la classification présentée dans la section précédente, le Tableau 1.1 résume les principales caractéristiques des différentes plateformes de RS utilisées.

Tableau 1.1 – Synthèse des caractéristiques de quelques RS

	Champ d'application		**Modèle de données**		**Plateforme**		**Réseau relationnel**	
Réseau Social	Loisir	Business	Centralisé	Décentralisé	Web	Cloud	Relationnel	Thématique
Facebook	✓		✓		✓	✓	✓	✓
MySpace	✓		✓		✓		✓	
Hi5	✓		✓		✓	✓	✓	
Flickr	✓		✓		✓	✓	✓	
LinkedIn		✓	✓		✓		✓	
Twitter	✓		✓		✓		✓	
YouTube	✓			✓	✓	✓		✓

Nous pouvons constater que la plupart des RS sont destinés au loisir et quelques uns uniquement sont utilisés pour le business comme LinkedIn. Nous remarquons qu'en pratique, plusieurs utilisateurs s'inscrivent à différents RS pour couvrir leurs besoins aussi bien en loisir qu'en business. Nous concluons alors que le développement d'un RS qui couvre les divers besoins est une bonne proposition qui peut attirer un grand nombre d'utilisateurs.

Comme montré par le tableau précédent, la plupart des RS existants utilisent une infrastructure centralisée où une seule unité de contrôle pour des données distribuées sur un nombre limité de serveurs. Nous observons une déviation de l'infrastructure centralisée à l'infrastructure distribuée [33]. En général, la décentralisation peut apporter des remèdes aux problèmes soulevés de la controverse dans le contexte de RS centralisés, comme la confidentialité et l'intégrité des informations personnelles [27, 34]. La décentralisation promet de meilleures performances, la tolérance aux pannes et la scalabilité en présence d'une base d'utilisateurs et d'applications en pleine expansion. La décentralisation de RS a été identifiée

comme un défi clé des recherches [33]. Cependant, le passage vers une architecture entièrement distribuée de RS n'est pas trivial. Cette migration donne lieu à de nombreuses questions de recherche concernant la mise en réseau et l'analyse des RS et des réseaux distribués.

Nous constatons aussi dans ce tableau que la plupart des RS sont fondés sur les relations sociales entre les utilisateurs (RS relationnels). Ainsi, l'échange de données sur les RS reste restreint aux groupes d'amis, ne respecte pas leurs intérêts et ne couvre pas tous les sujets d'actualités en temps réel. Ceci dépend des partages faits par les utilisateurs, la taille des groupes d'amis et les intersections entre eux. De plus, un utilisateur peut fréquemment consulter des contenus qui ne lui sont pas appropriés, ce qui cause une perte de temps, augmente inutilement la consommation de la bande passante et dégrade la qualité de service des RS.

Pour les RS qui sont thématiques, le type d'échange ne permet pas encore de bien exprimer les intentions des utilisateurs et du coup les mêmes limites persistent. La considération des intérêts détaillés des utilisateurs s'avère inévitable afin d'améliorer la performance du réseau notamment du point de vue de la qualité du service offert et la bande passante consommée. En effet, l'échange de données selon les intérêts réduit considérablement le trafic en éliminant les échanges inutiles.

Nous cherchons alors à ce que les contenus reçus par un utilisateur répondent à ses besoins en temps réel de façon que dès qu'un élément soit publié sur le réseau, il serait envoyé à toute(s) personne(s) intéressée(s). L'échange de contenus ne suit pas alors les relations d'amitié seulement mais se fait plutôt selon les intérêts exprimés par les participants. L'intérêt des participants est exprimé par eux-mêmes, peut couvrir une ou plusieurs thématiques selon une composition bien déterminée et peut aussi être mis à jour à tout moment. Ceci les aide à retrouver les contenus bénéfiques et en même temps à organiser leur vie sociale et professionnelle.

1.8 Motivation pour le développement d'un réseau social distribué (RSD)

A ce stade, il est indispensable de se demander, pourquoi utiliser une infrastructure distribuée pour supporter les RS, quand l'autre architecture (centralisée) fonctionne bien.

Un des avantages immédiats du modèle décentralisé est assez trivial : il n'est pas centralisé et il n'appartient pas à une seule entité. Nous pouvons aussi donner l'argument traditionnel que le distribué s'adapte parfaitement au besoin de la scalabilité, car un nombre croissant d'utilisateurs entraîne naturellement une consommation croissante de ressources réseaux.

Une autre incitation pour que les utilisateurs adoptent ce modèle est de surpasser les contraintes imposées par les fournisseurs de services telles que l'espace de stockage offert limité, les frais d'inscriptions, etc. Cette population devrait être la première qui adopte les RS s'appuyant sur une architecture pair-à-pair (P2P) avec un overlay formé par les équipements des intervenants eux-mêmes. En effet, les informations fournies par les utilisateurs doivent être partagées et routées par les équipements des utilisateurs eux-mêmes sans faire intervenir une tierce partie. Ainsi, les utilisateurs avec leurs équipements jouent le rôle de fournisseurs, consommateurs et routeurs d'informations. Par conséquent, nous aurons une augmentation proportionnelle entre le nombre d'utilisateurs avec leurs équipements et la quantité de données partagées sur le RS. Une infrastructure P2P peut profiter aussi du réseau formé par les relations sociales et la proximité géographique entre utilisateurs. Une approche P2P faisant intervenir les ressources des utilisateurs paraît alors efficace pour atteindre les objectifs de fiabilité et de passage à l'échelle en même temps.

1.9 RSD académiques

Après l'analyse des différentes propriétés de RS déployés sur Internet, nous nous concentrons principalement sur les nouvelles architectures proposées pour les RS distribués (RSD) [23, 35, 36]. L'apparition de RSD a créé de nouvelles perspectives et défis pour la mise en réseau [33, 37]. L'objectif est de construire des services de mise en réseau fondamentaux qui forment la base des services et des applications des RSD. Ceci nécessite une étude des nouvelles architectures, algorithmes et protocoles pour la mise en réseau de pairs de manière à fournir un environnement d'exécution et la fonctionnalité de communication de base requis pour les services et les applications de RSD.

Finalement, il est également important de comprendre comment le *workload* des RSD agit sur le trafic Internet vu son importance pour la conception de l'infrastructure Internet de la prochaine génération et des systèmes de distribution de contenus (par exemple, les CDN). Bien que les RSD contribuent beaucoup moins que les applications P2P en termes d'octets,

les RSD peuvent ajouter des fonctionnalités qui augmentent considérablement l'utilisation de la bande passante [33]. Vu cette augmentation excessive du trafic (cas où une vidéo devient populaire sur un site de RS), l'exploration des architectures proposées pour les RSD devient primordiale.

Dans la suite, nous présentons et expliquons brièvement les différentes approches existantes dans la littérature pour mettre en œuvre les RS en ligne décentralisés. Ces approches ont fait le sujet de propositions académiques non testées et non déployées sur Internet.

1.9.1 PeerSoN

PeerSoN [2] est l'un des premiers prototypes de RS distribués. Ce prototype a été proposé sur la base des idées développées dans [38], où les auteurs discutent des défis de décentralisation de RS et fournissent des solutions préliminaires pour les adresser. L'implantation de ce prototype prend en charge la procédure d'authentification des utilisateurs et le service de partage de fichiers entre eux.

PeerSoN propose une architecture à deux niveaux qui sépare les données de l'infrastructure de contrôle (voir Figure 1.4). Le niveau en bas englobe les utilisateurs et leurs données. Les utilisateurs peuvent stocker leurs données sur leurs propres systèmes de stockage et échanger directement des données entre eux-mêmes. Le niveau supérieur est une couche DHT (OpenDHT) qui offre des services de recherche (comme recherche d'amis, recherche de contenus, etc.), un service de stockage de métadonnées des utilisateurs et qui conserve les données pendant la période de déconnexion des utilisateurs. La décentralisation permet de se débarrasser du contrôle fait par une seule entité. Cependant, le service de recherche ne stocke pas les données et les contenus privés des utilisateurs mais il stocke seulement les métadonnées nécessaires pour traiter les requêtes de recherche.

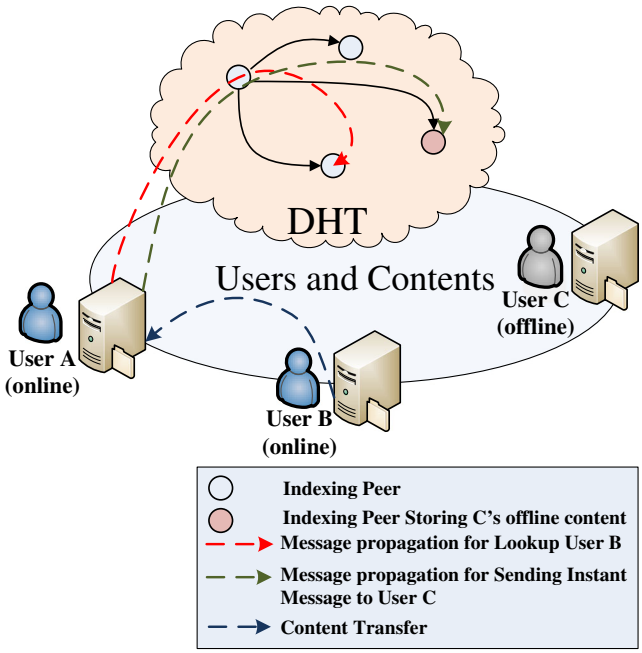


Figure 1.4 – Architecture de PeerSoN [2]

1.9.2 Safebook

Safebook [3] propose une architecture à trois niveaux pour un RSD mettant au point la confidentialité, l'intégrité et la disponibilité de données. Comme montré par la Figure 1.5, le niveau en bas présente les utilisateurs avec les relations sociales entre eux. Ce niveau est responsable du stockage des données, de la disponibilité des données et de la confidentialité des communications. Le deuxième niveau est une couche P2P superposée avec la couche des relations sociales et qui offre le service de recherche de données. Le réseau Internet se trouvant dans le haut niveau de l'architecture fournit des services de communication et de transport pour Safebook.

Safebook est un RSD dont l'objectif principal consiste à protéger la vie privée des utilisateurs. Il tient compte du comportement anormal ou erroné des fournisseurs de services, ainsi que des entités malveillantes externes qui pourraient intercepter ou modifier des données sur la couche réseau.

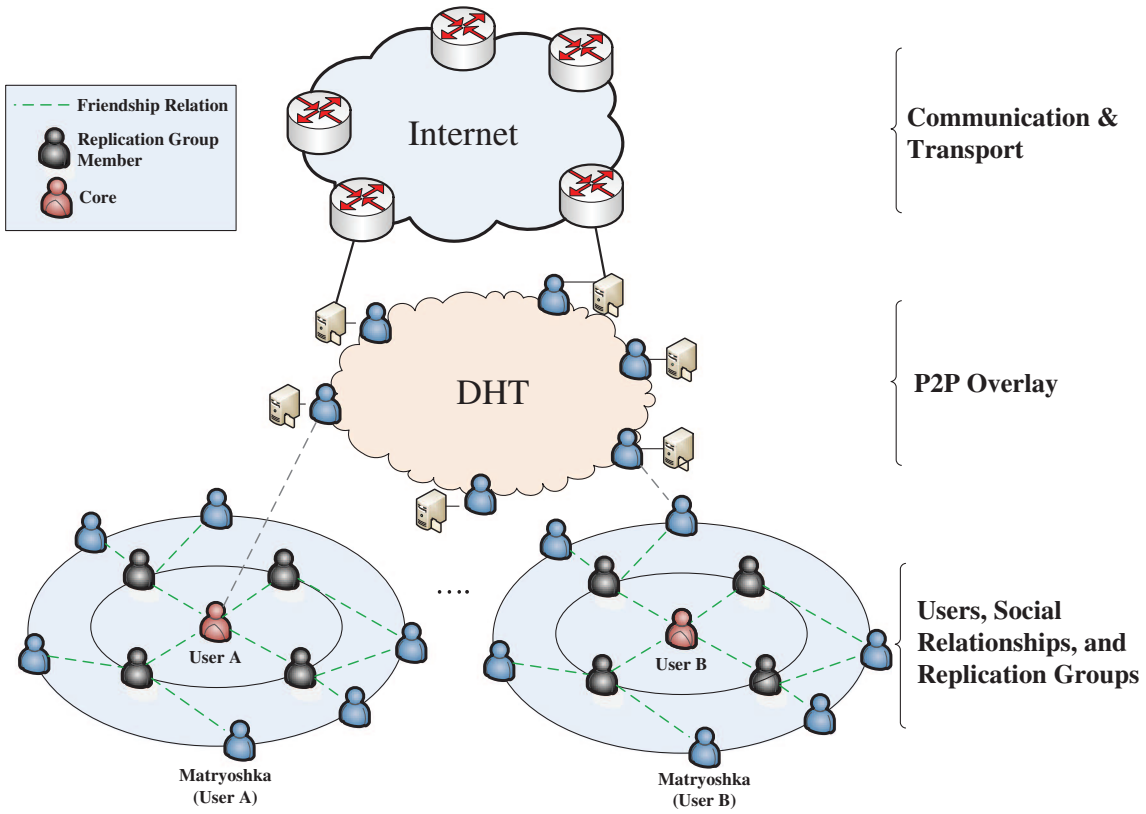


Figure 1.5 – Architecture de Safebook [3]

1.9.3 PrPl

PrPl [4] est un RSD permettant le contrôle de la confidentialité des données et fournit aussi une API pour le développement d'applications sociales qui peuvent fonctionner dans différents domaines. PrPl permet aussi aux utilisateurs de stocker leur données (photos, musiques à titre d'exemples) dans des équipements de leur choix. Le stockage distribué des données d'un utilisateur est fédéré par un service pour chaque utilisateur nommé Personal Cloud Butlers. Ce service garde une trace de l'emplacement du contenu et donne une image système unique du stockage sous-jacent distribué à d'autres utilisateurs (Figure 1.6). Ce service est également responsable d'assurer la confidentialité des données, le stockage des données privées d'un utilisateur, le maintien de la liste des amis (à savoir les relations de confiance) et de communiquer avec les services Butlers des autres utilisateurs pour accéder à leurs données.

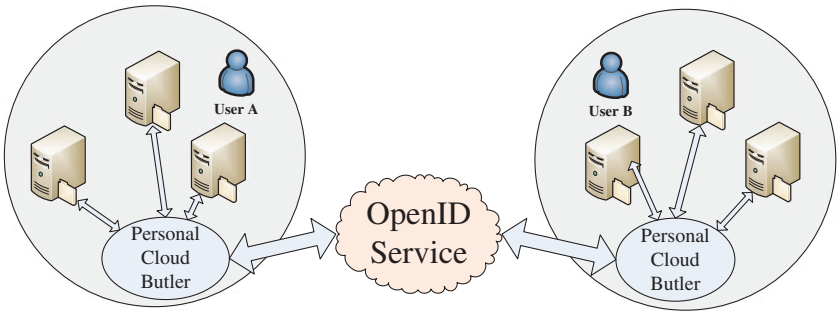


Figure 1.6 – Architecture de PrPl [4]

PrPl repose sur le service OpenID [39] pour la gestion des identités. OpenID est un système décentralisé permettant aux utilisateurs de s'authentifier auprès de plusieurs sites avec le même identifiant OpenID. Il est responsable de l'authentification des services Butlers pour le contrôle d'accès. Le service Butler fournit à son propriétaire différents droits d'accès pour ses données. Ainsi, il devient possible de permettre/refuser les demandes d'accès entrants à un contenu individuel pour les utilisateurs. PrPl suppose que le service Butler et le stockage des données personnelles peuvent être déployés sur des équipements à disponibilité élevée tels que les home gateways, les terminaux, les consoles de jeux, etc. Selon cette supposition, PrPl ne traite pas le problème de disponibilité de données et ne fait ni la réplication de données ni la mise en cache pour assurer la disponibilité du service Butler et du système de stockage de l'utilisateur.
Pour les utilisateurs ayant un grand nombre d'amis, il y aura un problème de scalabilité à cause de la surcharge de leur service Butler étant donné que le service Butler est le seul point de contact pour un utilisateur avec ses amis.

1.9.4 SuperNova

SuperNova propose une architecture de RSD fondée sur le concept de super pairs [5]. Il permet aux utilisateurs d'être une partie du RS et de partager leurs données tout en restant les propriétaires de ces données. En outre, les utilisateurs peuvent imposer différents droits d'accès (tels que public, privé ou protégé) à leurs données. Dans SuperNova, les super pairs participent activement à la formation de l'infrastructure de contrôle du RS. Les utilisateurs de SuperNova peuvent stocker leurs contenus sur leurs propres machines. Cependant, le système de stockage d'un utilisateur pourrait ne pas être toujours disponible (l'exemple

d'un téléphone portable qui ne peut pas avoir une connectivité permanente 24/7 à Internet). Pour faire face à ce problème, les utilisateurs peuvent répliquer leurs contenus à un ensemble d'utilisateurs connus comme magasiniers (*storekeepers*) comme montré par la Figure 1.7. Les super pairs sont responsables de suivre le modèle de disponibilité des utilisateurs qui leur sont assignés. Un super pair permet à ses utilisateurs de choisir leurs storekeepers en leur fournissant le modèle de disponibilité des autres utilisateurs. Un utilisateur sélectionne un autre utilisateur comme storekeeper quand il obtient son accord. Pendant le temps de disponibilité d'un utilisateur, des mises à jour de son contenu (par exemple, des commentaires sur les photos, vidéos, etc.) sont transmises à son super pair seulement. Les répliques sur les storekeepers sont mises à jour lorsque l'utilisateur est hors ligne et elles gèrent les mises à jour ultérieures jusqu'à ce que l'utilisateur se reconnecte une autre fois. Pour assurer la confidentialité des données, ces dernières sont cryptées sur les répliques. Quand un nouveau utilisateur rejoint le réseau, son super pair lui fournit un service de cache temporaire jusqu'à ce qu'il forme son groupe de répliques.

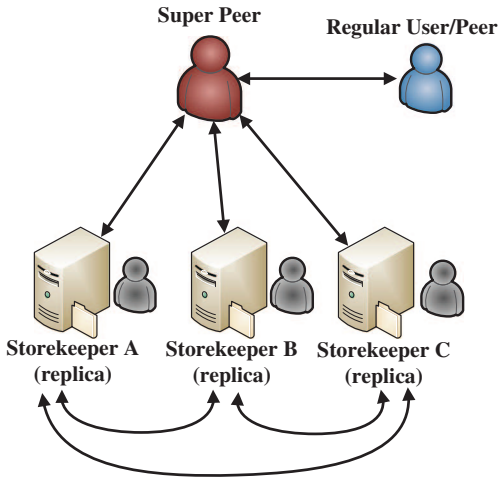


Figure 1.7 – Architecture de SuperNova [5]

Les supers pairs sont les entités de base pour l'architecture de SuperNova. Ils sont responsables de fournir le service de recherche (retrouver les storekeepers et rechercher les amis), le service de stockage et de gestion des comptes, le service de recommandation aux sous-ensembles d'utilisateurs et de maintenir leurs répliques.

1.9.5 Vis-á-Vis

Vis-á-Vis fournit un service de RS décentralisé [6]. Il utilise les services du Cloud Computing tels qu'Amazon EC2 au lieu d'utiliser les postes de travail personnels pour atteindre une haute disponibilité des services aux utilisateurs. Les machines virtuelles du Cloud, appelées serveurs individuels virtuels (*Virtual Individual Servers* VISs), fournissent des services de stockage et de calcul pour les utilisateurs. La Figure 1.8 montre l'architecture de Vis-á-Vis. Les utilisateurs mémorisent leurs données personnelles dans les VIS de confiance sans cryptage. Un utilisateur peut devenir membre d'un ou de plusieurs groupes et partage la localisation géographique avec les membres du groupe. Un groupe est l'unité du domaine autoritaire dans Vis-á-Vis, autrement un groupe est la plus petite unité sous le contrôle d'une seule autorité. Les utilisateurs ont la liberté d'appartenir à différents groupes. Ils interagissent avec les groupes à travers leurs VIS. Les groupes sont créés et gérés par les utilisateurs.
Par ailleurs, un arbre de localisation est formé pour mémoriser l'information sur chaque subdivision d'emplacement couverte par le groupe. Chaque nœud intermédiaire de cet arbre représente une subdivision d'une région. Le VIS d'un utilisateur peut être une feuille de l'arbre de localisation selon l'emplacement partagé avec le groupe. Un coordinateur (un VIS) est affecté à chaque nœud intermédiaire de l'arbre de localisation pour le traitement des requêtes de recherche destinées aux sous-arbres en dessous. Le coordinateur d'un nœud est élu par tous les coordinateurs de ses descendants en utilisant un protocole de consensus distribué tel que Paxos pour une période de contrat prédéfinie. Zookeeper [40] est également utilisé en tant que service de coordination distribué. Pour étendre la durée de vie, chaque coordinateur émet des messages de renouvellement de contrat périodiquement en utilisant le multicast. Chaque groupe dispose également d'un service d'adhésion pour soutenir la politique d'accès au groupe. De plus, le service d'adhésion mémorise une référence à l'arbre de localisation du groupe.

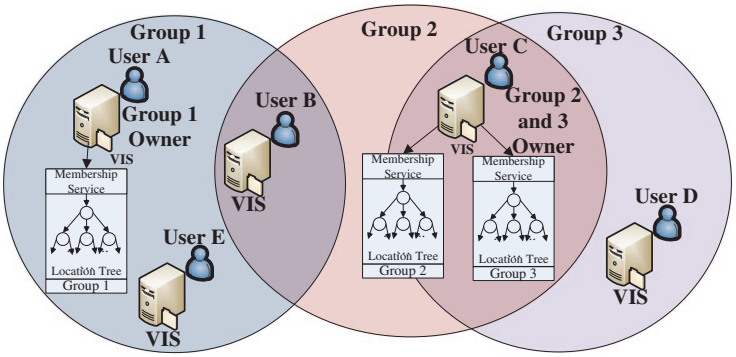


Figure 1.8 – Architecture de Vis-á-Vis [6]

1.9.6 Cachet

Cachet [7] est un RS purement décentralisé où les utilisateurs collaborent les uns avec les autres pour stocker leurs contenus sans aucun service centralisé. Il est construit au dessus de DECENT [41] et il améliore son temps de latence élevé du flux de production de DECENT. La Figure 1.9 montre son architecture au moment de la diffusion de contenus. Dans DECENT, les utilisateurs du réseau présentent des conteneurs d'objets et forment alors une DHT pour stocker des objets. Un objet conteneur est l'unité de contrôle d'accès dans le réseau. Cet objet stocke le contenu réel et peut contenir des références à d'autres objets conteneurs. Le contenu peut parfois prendre la forme de références à d'autres objets conteneurs. Un exemple typique de ce concept est l'album photo dans lequel l'objet conteneur stocke les références des contenants de l'ensemble de ses photos. Tous les objets conteneurs utilisent les identificateurs générés aléatoirement pour le placement dans DHT et sont stockés après chiffrement dans les nœuds de stockage.

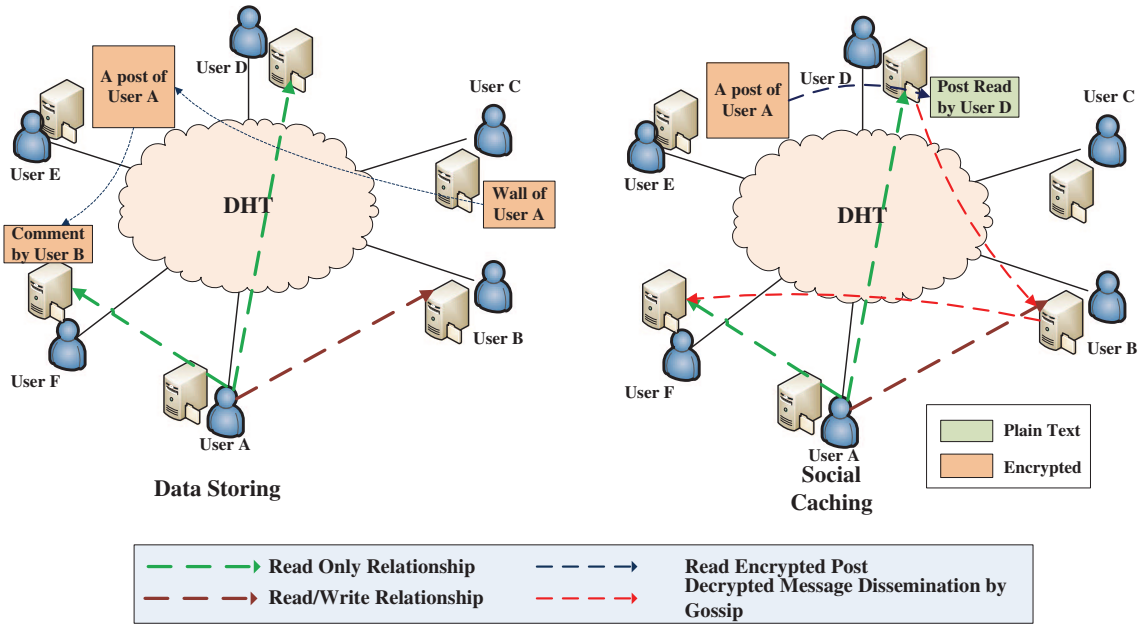


Figure 1.9 – Architecture de Cachet [7]

Les utilisateurs de Cachet choisissent une politique d'accès aux objets conteneurs telle que lecture, écriture et ajout, mémorisée avec les métadonnées de l'objet pour dicter le mécanisme de contrôle d'accès aux objets. Ces politiques peuvent dépendre de l'identité de l'utilisateur ou du groupe depuis lequel le contenu est accessible. Un utilisateur génère plusieurs paires de clés de chiffrement pour chacune des politiques. Quand une relation se forme entre deux utilisateurs, le premier utilisateur (utilisateur A) fournit au deuxième (utilisateur B) une clé de cryptage en fonction de la politique de cette relation. Cette clé de chiffrement régit l'accès de l'utilisateur B à la totalité ou à un sous-ensemble des données de l'utilisateur A dans le réseau. L'utilisateur B, à son tour, fournit également une clé à l'utilisateur A pour accéder à son contenu. Cependant, les capacités des clés de chiffrement échangées par les deux utilisateurs peuvent ne pas être les mêmes, ce qui entraîne une relation asymétrique entre les utilisateurs. Un utilisateur ayant l'autorisation peut localiser les données d'un ami sur DHT et les décrypter avec la clé fournie par le propriétaire. Tout utilisateur peut ajouter aux données de ses amis s'il a l'autorisation pour le faire. Les données ajoutées sont stockées dans un objet conteneur distinct sur DHT et la référence à cet objet est insérée dans l'objet conteneur de la donnée lue précédemment. Cet objet conteneur nouvellement créé est également crypté avec une clé privée de son propriétaire et ne peut être lu que par les utilisateurs autorisés.

La technique de cryptage dans l'architecture de base de Cachet nécessite des opérations de décryptage coûteuses des différents contenus du contact social d'un utilisateur lors de la génération de nouveaux flux de données. Pour améliorer le temps de réponse, les auteurs ajoutent à l'architecture de base de Cachet une entité de mise en cache sociale (social caching). Ainsi, chaque nœud de l'utilisateur maintient une connexion sécurisée avec l'ensemble de ses contacts sociaux en ligne. Tout nœud qui satisfait la politique fondée sur les attributs d'un contenu spécifique peut mettre en cache ce contenu. De plus, le nœud peut fournir cette cache et les données en clair aux autres nœuds qui satisfont également la politique. Cette dernière caractéristique accélère également la récupération de contenus lorsque le propriétaire du contenu est déconnecté. Pour maintenir une connexion directe avec les utilisateurs en ligne, une liste appelée *liste de présence* est stockée sur DHT [33]. La mise à jour de cette liste se fait exclusivement par le propriétaire chaque fois qu'il rejoint ou qu'il quitte Cachet. Pour éviter le cryptage, les objets de présence sont également mis en cache et diffusés en utilisant un protocole de dissémination fondé sur le "gossip".

1.9.7 Microblogs en P2P

Un certain nombre d'architectures ont été proposées dans la littérature pour les microblogs décentralisés ayant des caractéristiques similaires à Twitter. Le premier projet de ce genre était FETHR (Featherweight Entangled Timelines over HTTP Requests) [8]. Plus tard, Cuckoo [9] et d'autres systèmes de microblogging décentralisés sont apparus comme étant une version améliorée de FETHR. Dans le reste de cette section, nous décrivons ces deux systèmes en détails.

1.9.7.1 FETHR

FETHR [8] propose une architecture décentralisée fondée sur le gossip (Figure 1.10). C'est la première tentative de proposer un RSD similaire à Twitter. La principale contribution de ce projet est un protocole de communication fondé sur HTTP qui permet aux utilisateurs de communiquer directement entre eux. FETHR supporte des fonctionnalités semblables à celles de Twitter telles que "following" et "tweeting".

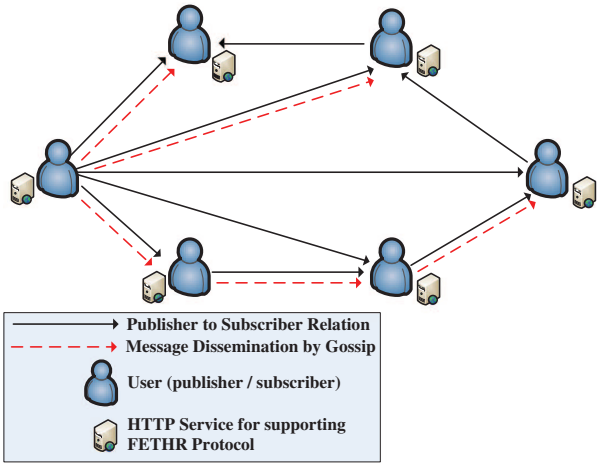


Figure 1.10 – Architecture du microblog décentralisé Fethr [8]

Un utilisateur de FETHR peut souscrire aux mises à jour des autres utilisateurs par simples messages HTTP (GET et POST). La publication d'un contenu suit un modèle push où les producteurs poussent leurs mises à jour directement à leurs souscrits. Pour atténuer la surcharge due à la propagation des mises à jour dans le réseau, FETHR propose de diffuser la mise à jour avec le protocole de gossip, où les mises à jour sont poussées à un sous-ensemble de souscrits qui, à leur tour, prennent la responsabilité de les pousser au reste du réseau. Les contenus envoyés à des souscrits augmentent la disponibilité des contenus, car ils ne peuvent être servis à partir de ces derniers que lorsque l'éditeur est hors ligne.

1.9.7.2 Cuckoo

Cuckoo [9] propose une architecture décentralisée et supporte des services semblables à Twitter comme, par exemple, les souscriptions et les publications de microblog. Contrairement à son prédécesseur FETHR, Cuckoo a une architecture structurée (Figure 1.11). Il utilise une couche structurée DHT pour assurer le service de recherche. En effet, un nouvel utilisateur ou un utilisateur déjà connecté peut utiliser DHT pour retrouver d'autres utilisateurs. Toutefois, ce processus de recherche ne dépend pas uniquement de la DHT, il suit plutôt une approche hybride. L'approche hybride combine la capacité de la DHT de garantir une recherche efficace d'objets rares et d'utiliser les inondations (*flooding*) pour une recherche rapide d'objets populaires. Cuckoo adopte une stratégie de *push* pour publier les microblogs. Les producteurs utilisent une stratégie de publication fondée sur *gossip*, semblable à celle

de FETHR. Les contenus poussés vers les abonnés agissent comme une réplique du contenu original et augmentent la visibilité du contenu original pendant la période hors-ligne du producteur.

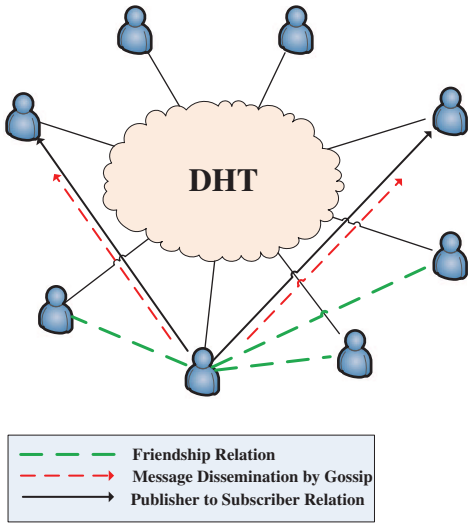


Figure 1.11 – Architecture du microblog décentralisé Cukoo [9]

La différence de base entre Cuckoo et son prédécesseur FETHR, est la DHT. FETHR ne fournit pas de détails sur la gestion des utilisateurs dans un système décentralisé. Cuckoo traite cette question en utilisant DHT. La DHT est formée par la participation directe de l'utilisateur du système, et donc, le fonctionnement du système ne repose pas sur un service centralisé et il n'est pas contrôlé par une autorité centrale.

1.9.8 Synthèse

Le Tableau 1.2 présente un récapitulatif des caractéristiques des RSD proposés par des travaux académiques.

Les RS proposent des architectures distribuées pour traiter essentiellement la confidentialité et l'intégrité des données. L'architecture la plus utilisée était le P2P structurée qui utilise DHT. Pour tous les RSD sauf Cachet, cette architecture fait recours à des serveurs étrangers soit pour stocker les données ou pour servir comme routeurs entre les utilisateurs.

Tableau 1.2 – Caractéristiques des RS distribués

Réseaux Sociaux	**Architecture**	**Partage de contenus multimédias**	**Disponibilité des données**	**Intérêt des utilisateurs**
PeerSoN	DHT	✓	-	simple
FETHR	non-structurée	-	réplication	simple
Safebook	DHT	✓	réplication	simple
PrPl	DHT	✓	-	-
Cuckoo	DHT	-	réplication	-
Vis-á-Vis	hiérarchique	-	-	-
SuperNova	super pair	✓	réplication	simple
Cachet	DHT	✓	réplication et mise en cache	-

Le problème de disponibilité de données sur les RSD est un sujet d'actualité où peu de solutions ont été proposées pour le résoudre puisqu'ils sont récemment apparus. En effet, la majorité des RSD s'intéressent plus à la confidentialité des données qu'à leur disponibilité. Le traitement de la disponibilité de données, si considéré, est souvent conduit par les utilisateurs (propriétaires des données) dans le but de garder la sécurité de l'information. Néanmoins, ceci ne garantit pas la disponibilité de données puisque les responsables des répliques peuvent avoir des taux de disponibilité faibles. En réalité, ces équipements responsables n'ont pas nécessairement une disponibilité élevée et ils peuvent même être défaillants. Par conséquent, assurer une disponibilité de 24/7 des contenus des utilisateurs stockés dans un cadre décentralisé reste un défi. De plus, nous ne retrouvons aucune solution dans la littérature qui traite la disponibilité de données de bout en bout afin d'assurer leur livraison sans perte.

Comme avec les RS présentés dans la première partie de ce chapitre, la majorité des RSD n'assurent le partage de données qu'entre les amis de groupes biens déterminés. Quelques RSD uniquement prennent en compte les intérêts des utilisateurs, mais ceci reste toujours restreint aux intérêts simples qui manquent encore d'expressivité pour répondre aux besoins exacts des utilisateurs. Prendre en considération les intérêts détaillés des utilisateurs est une obligation afin d'améliorer la performance du réseau notamment du point de vue service offert et bande passante consommée. En effet, l'échange de données selon les intérêts réduit

considérablement le trafic en éliminant les échanges inutiles. Nous cherchons alors à ce que les contenus reçus par les utilisateurs doivent répondre à leurs besoins en temps réel de façon que dès qu'un élément soit publié sur le réseau, il sera envoyé à toute(s) personne(s) intéressée(s). L'échange de contenus ne suit pas alors les relations d'amitié seulement mais se fait aussi selon les intérêts exprimés par les participants. L'intérêt des participants est exprimé par eux-mêmes, peut couvrir une ou plusieurs thématiques selon une composition bien déterminée et peut aussi être mis à jour à tout moment. Ceci les aide à retrouver les contenus bénéfiques pour eux et en même temps à organiser leurs vies sociales et professionnelles.

1.10 Conclusion

Dans une première partie, nous avons étudié les RS. Nous avons trouvé que l'échange de contenus entre les utilisateurs sur les RS reste toujours limité par les relations d'amitié entre eux. Ceci dépend des partages faits par les utilisateurs, la taille des groupes d'amis et leurs intersections. Ce type de RS résulte en un trafic important et souvent inutile sur le réseau. En effet, un utilisateur peut fréquemment consulter des contenus qui ne lui sont pas appropriés ce qui engendre une perte de temps et de bande passante et rend les RS ennuyants.
Cette étude a montré aussi qu'il y a une tendance à développer des RSD pour des raisons de sécurité et mise à l'échelle. De nouvelles approches ont été proposées dans ce contexte mais qui souffrent encore de quelques limites telles que l'expression des intentions des utilisateurs, la disponibilité de données et les coûts de ressources pour ces RSD.

Nous proposons dans la suite de développer un RSD en P2P permettant une consommation sélective de contenus composites entre des utilisateurs distants. L'échange de données par ce RSD repose sur les intentions composites exprimées par les utilisateurs.

Chapitre

2 MULUS : un RSD pour une consommation sélective de contenus composites

2.1 Introduction

Dans le chapitre précédent, nous avons étudié les RS déployés sur Internet ainsi que les nouvelles approches proposées pour leur décentralisation. Nous avons conclu que ces RS présentent encore plusieurs lacunes au niveau de leur architecture, la disponibilité de données, la livraison sans perte et l'expression des intentions des utilisateurs devant la grande quantité de données produites sur Internet. De plus, les besoins d'un seul utilisateur peuvent exiger l'utilisation d'un ou plusieurs RS puisque ces derniers peuvent êtres de domaines d'applications différents (loisir, business, etc.). D'autre part, pour avoir un meilleur service avec plus de fonctionnalités, certains RS exigent des frais d'inscription auprès des abonnés. Pour mieux répondre aux préférences des utilisateurs, nous proposons une architecture qui adapte la livraison des données en fonction des intérêts des utilisateurs. Cela permet, entre autres, de réduire le trafic sur le réseau en éliminant les partages inutiles.
Les systèmes publier/souscrire confortent les intérêts des utilisateurs. Il est alors intéressant de les mettre en place afin de résoudre la problématique de cette thèse. Nous développons alors une nouvelle plateforme que nous avons nommé MULUS, fondé sur les systèmes publier/souscrire pour supporter la livraison en P2P de contenus composites entre les utilisateurs.

Dans ce chapitre, nous commençons par présenter les concepts de base des réseaux P2P ainsi que les systèmes publier/souscrire et leurs architectures pour en sélectionner le meilleur choix du système de base pour notre plateforme. Puis, nous présentons l'approche proposée et nous décomposons la problématique pour pouvoir la traiter dans les chapitres suivants.

2.2 Concepts de base

Dans cette section, nous présentons les concepts de base relatifs aux différents outils que nous allons utiliser dans notre approche à savoir les systèmes P2P, les systèmes publier/-souscrire et les ontologies.

2.2.1 Système P2P

Les systèmes de communication de nos jours deviennent beaucoup plus complexes que les systèmes distribués classiques vu leur organisation très décentralisée et l'absence de gestion de l'état global du système. Il y a eu beaucoup d'intérêts aux overlays P2P puisqu'ils fournissent un bon substrat pour le partage de données à large échelle, la distribution de contenus et le multicast à la couche applicative. Le plus grand apport de ces systèmes P2P est qu'ils ne nécessitent pas de gigantesques serveurs à installer ou à connecter pour servir comme intermédiaires de stockage ou de routage de données entre utilisateurs. Cet avantage libère les utilisateurs de ces systèmes des coûts supplémentaires des services. En effet, au sein d'un système P2P, chaque pair, cherchant un service, doit participer en donnant accès à une "partie" de ses ressources. Plus il participe avec de bonnes ressources, plus le service en totalité est de meilleure valeur. Tous les pairs jouent des rôles équivalents : utilisateur, serveur, routeur ou/et hôte. Un pair représente un serveur s'il offre des ressources, un utilisateur quand il consomme des ressources disponibles, un routeur quand il propage les requêtes reçues aux autres pairs du système et un hôte de sources de données lorsqu'il fournit ses propres données aux autres nœuds.

Dans la suite, nous détaillons la définition d'un système P2P et ses caractéristiques, puis nous nous concentrons sur la DHT (Distributed Hash Table) comme système P2P structuré.

2.2.1.1 Définition d'un système pair-à-pair

Un aperçu rapide dans la littérature révèle un nombre considérable de définitions du "Système P2P".

Une définition, plus stricte du système P2P, se réfère à des systèmes totalement distribués, dans lesquels tous les nœuds sont complètement équivalents en termes de fonctionnalité et de tâches qu'ils accomplissent. Cette définition ne parvient pas à couvrir, par exemple, les systèmes qui utilisent la notion de "super-nodes" (nœuds qui fonctionnent comme mini-serveurs

localisés et assignées dynamiquement) tels que Kazaa [42], qui sont, toutefois, largement acceptés comme systèmes P2P, ou des systèmes qui s'appuient sur une infrastructure de serveurs centralisée pour un sous-ensemble de tâches non essentielles.

Selon une définition plus large et largement acceptée dans Shirky [43], "le P2P est une classe d'applications profitant des ressources (stockage, calcul, contenu, présence humaine) disponibles sur des nœuds connectés à l'internet". Cependant, cette définition englobe les systèmes qui dépendent entièrement de serveurs centralisés pour leur fonctionnement (tels que SETI@home [1], divers systèmes de messagerie instantanée, ou encore le fameux Napster), ainsi que de diverses applications du domaine de Grid Computing.

Les auteurs de [44] ont donné la définition suivante : "les systèmes P2P sont des systèmes constitués de nœuds interconnectés et capables de s'auto-organiser selon la topologie du réseau, dans le but de partager des ressources telles que le contenu, les cycles de CPU, le stockage et la bande passante. Ils sont capables de s'adapter à des défaillances (pannes de nœuds) et de percevoir des informations concernant les autres pairs tout en maintenant une connectivité et une performance acceptables, sans recours à un intermédiaire comme un serveur global ou une autorité centralisée".

2.2.1.2 Caractéristiques des systèmes P2P

Les systèmes P2P sont très populaires par rapport aux systèmes fondés sur le modèle client/serveur grâce à leurs caractéristiques avantageuses. Ils visent à assurer les caractéristiques suivantes :

- scalabilité : il s'agit d'interconnecter et de faire coopérer un grand nombre de pairs (des milliers voire des millions) pour partager des ressources, tout en assurant une bonne performance de tout le système. Autrement, un système P2P doit maintenir sa performance indépendamment du nombre de pairs interconnectés, du volume de données partagées et du nombre de messages échangés, tout en partageant ses ressources via un réseau largement distribué.

- autonomie des nœuds : chaque nœud participant gère ses ressources et la partie de ses données à partager d'une manière autonome. Il se connecte ou/et se déconnecte à tout moment. Il possède aussi l'autonomie de gérer sa capacité de stockage et sa puissance de calcul. Il peut choisir la méthode de conception de bases de données et le système de gestion

1. http ://setiathome.ssl.berkeley.edu/

de base de données adéquats au domaine d'application.

- auto-adaptation à l'environnement dynamique : l'instabilité des nœuds cause la disparition de ressources (lors de la déconnexion d'un ou de plusieurs nœuds) et l'apparition de nouvelles ressources (lors de la connexion d'un ou de plusieurs nouveaux nœuds). De ce fait, ces systèmes doivent bien gérer toutes les ressources fortement variables. La déconnexion d'un nœud du système (ou sa panne) doit être tolérée pour ne pas causer l'échec du système. Elle peut avoir un "léger" impact sur la performance du système en totalité.

- auto-configuration : un système P2P doit être un environnement ouvert qui permet aux utilisateurs de connecter leurs nœuds au système sans passer par une administration centralisée. Pour se connecter au système, il faut avoir un point d'accès à Internet et connaître un autre nœud déjà connecté. De plus, les utilisateurs n'ont pas besoin d'une infrastructure coûteuse puisque ce type de système est généralement déployé sur Internet.

- décentralisation : un système P2P permet d'éviter la centralisation du contrôle par le fait que chaque nœud est responsable de la gestion de ses propres ressources. Il peut fonctionner donc sans administration centralisée, ce qui permet de limiter les goulets d'étranglements et de faire face aux pannes et aux défaillances du système.

- hétérogénéité : les systèmes P2P doivent avoir des techniques appropriées pour adresser les défis liés à l'hétérogénéité des ressources appartenant aux différents nœuds constituant le système.

2.2.1.3 Classifications des systèmes P2P

Dans un système P2P, les nœuds connectés forment un overlay, situé au dessus du réseau physique Internet. L'architecture d'un système P2P décrit alors la structure du réseau virtuel formé par les nœuds. Cette architecture influence directement la performance du système. Elle permet aussi une distinction de trois classes de système P2P : système P2P non-structuré, système P2P structuré et système avec super-pairs.

1. Système avec super-pairs

Cette approche consiste à attribuer des rôles différents aux nœuds, que ce soit le rôle de serveur (super-pair) ou le rôle de client, selon leurs capacités en termes de disponibilité, bande passante, capacité de stockage, etc. Ainsi, cette approche propose une architecture hybride entre l'architecture P2P et l'architecture client/serveur pour profiter de leurs avantages. Elle attribue le rôle de serveurs aux nœuds puissants et le rôle de clients au reste des nœuds.

Les nœuds serveurs sont appelés super-pairs (*super-node* en anglais). Ils se connectent entre eux selon une architecture P2P structurée ou non structurée (expliquée ultérieurement). Ils sont responsables de la gestion des ressources partagées sur le réseau, du contrôle d'accès, de l'indexation des données et de l'évaluation des requêtes.
Après avoir rejoint le réseau du système P2P, le pair participant informe le serveur auquel il se connecte, sur le contenu qu'il stocke. Les emplacements des données sont donc mémorisés et indexés par les serveurs centraux qui sont responsables de recevoir les requêtes et leur répondre.

Un système fondé sur ce type de réseau est plus sensible aux pannes que les autres types de systèmes P2P. Éventuellement, il expose les serveurs centraux à l'attaque, à la défaillance ainsi qu'au goulot d'étranglement. En effet, lorsqu'un serveur tombe en panne, ses utilisateurs deviennent isolés du reste du système et privés de ses services. Pour faire face à ce problème, les super-pairs peuvent être dynamiquement changés si une panne ait lieu. Le choix d'un nouveau super-pair peut se faire selon la capacité de calcul, la disponibilité ou la bande passante.

2. Système P2P non structuré

Pour cette deuxième génération, il n'y a aucune connaissance *a priori* sur la topologie du réseau. Les pairs rejoignent le réseau sans règles spécifiques et sans suivre aucune structure, en se connectant à un ou plusieurs nœuds, dits voisins. Une ressource peut prendre un certain temps pour l'opération de recherche, car il n'y a pas de relation entre le nom des ressources et leurs emplacements. En effet, la recherche de données se fait par inondation de la requête à tous les pairs participant au système jusqu'à ce que la donnée désirée soit trouvée. Si un pair reçoit une requête, il diffuse ce message aux autres pairs jusqu'à ce que le TTL (Time to Live) soit atteint, tel que le cas de Gnutella [10]. L'avantage du système P2P non structuré est que la taille du réseau peut être infinie et le coût de stockage est de l'ordre de O(1) puisque les données ne sont enregistrées que dans les pairs fournissant les données. En outre, c'est une technique adéquate pour les requêtes complexes. Cependant, la complexité de recherche est de l'ordre de $O(N^2)$ et les résultats de recherche ne sont pas garantis si la durée de vie TTL est limitée et si le nombre de nœuds est assez grand.

Plusieurs protocoles ont été développés et destinés pour les systèmes P2P non structurés, tels que le protocole Freenet [45] et le protocole Gnutella [10]. Ce dernier est présenté à titre d'exemple dans la Figure 2.1.

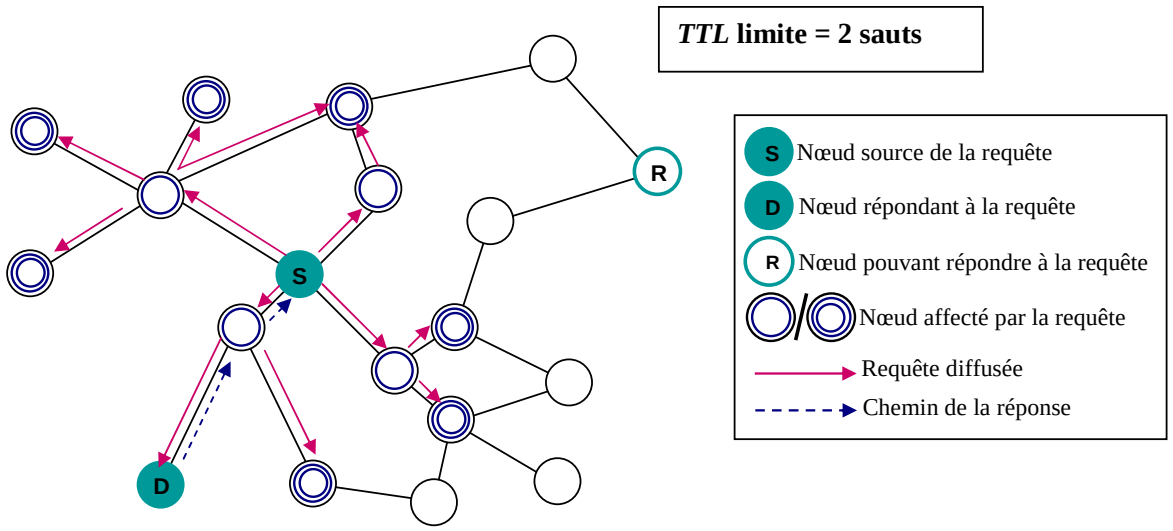


Figure 2.1 – Principe de fonctionnement du protocole Gnutella [10]

Les autres avantages de cette classe sont : la mise en œuvre, la simplicité, la recherche par mot clé ou par requête complexe et le support des environnements dynamiques. Les limites majeures de cette classe sont la consommation élevée de la bande passante et le problème de passage à l'échelle.

3. Système P2P structuré

Pour résoudre les problèmes qui se sont posés dans les systèmes non structurés, il y a eu l'émergence des systèmes P2P structurés fondés sur les tables de hachage distribuées (DHT). Contrairement aux systèmes P2P non structurés, chaque pair de la DHT a une connaissance locale du réseau. De ce fait, la localisation d'un objet se produit en routant une requête au pair voisin le rapprochant le plus de la cible. Un système P2P structuré permet de trouver n'importe quel objet s'il existe en $O(log\ n)$ sauts, où n est le nombre de pairs dans le réseau. Il supporte généralement un routage déterministe et une recherche structurée.

La distribution des ressources et la structure du réseau sont corrélées. En effet, une DHT définit un espace d'identifiants pour assigner des identificateurs aux pairs participants et des clés pour les différents blocs de données, grâce à l'utilisation d'une fonction de hachage telle que SHA-1. Tous les nœuds du réseau sont organisés selon une architecture structurée telle qu'en anneau [46], un espace d-dimensionnel [47], un arbre binaire [48], etc. Les ressources sont déposées au niveau des nœuds en fonction de leurs clés. Ainsi, chaque nœud est responsable des clés voisines de son identification. Par la suite, pour localiser une donnée, il

faut déterminer le nœud responsable de la clé de cette ressource. Le nœud responsable est le nœud ayant un identifiant numériquement plus proche de la clé de la donnée recherchée.

Une DHT utilise la fonction de localisation dans deux opérations importantes. D'une part, pour insérer une donnée de valeur v_1 dans le réseau, il est nécessaire tout d'abord de dériver sa clé k_1 en utilisant une fonction de hachage, puis d'insérer le couple (k_1, v_1) au niveau du pair responsable de cette clé k_1. D'autre part, l'accès à une donnée se fait à partir de sa clé, en routant une requête au pair voisin qui numériquement la rapproche le plus de la destination désirée. Le principe de routage DHT est montré par la Figure 2.2.

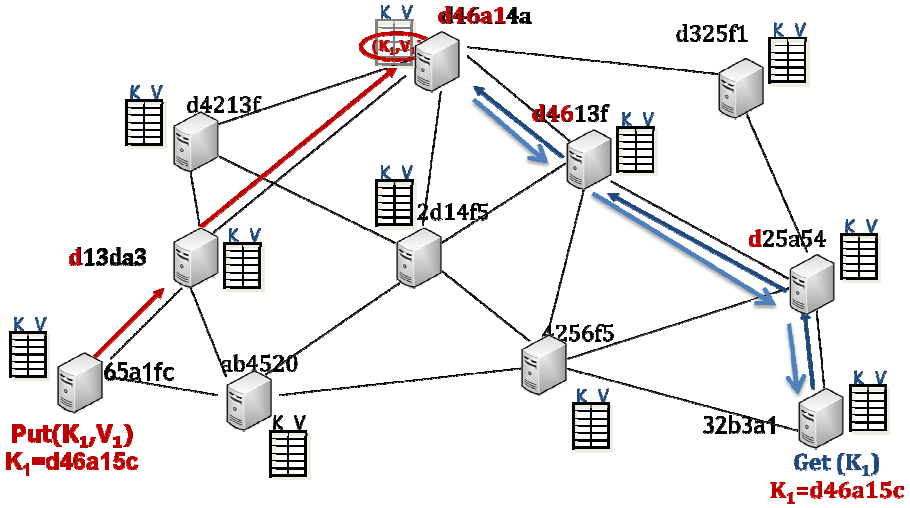


Figure 2.2 – Espace d'identifiants pour une DHT

Plusieurs protocoles ont adopté les principes de base de DHT, comme Chord [46], CAN [47], Pastry [19], Tapestry [49], etc. Ces protocoles se distinguent principalement par leurs stratégies d'identification et de routage. Trois protocoles de DHT ont suscité l'attention dans la communauté de recherche : Chord, CAN et Pastry.

2.2.2 Caractéristiques des tables de hachage distribuées (DHT)

Il est certain que les DHT ont des propriétés pertinentes et elles ont résolu plusieurs problèmes révélés par d'autres systèmes distribués. Elles permettent d'outrepasser certaines limites, telles que l'inadaptation aux changements fréquents et aléatoires de la topologie et le dynamisme fort des nœuds. Nous présentons dans ce qui suit les caractéristiques des DHT.

a. Faible diamètre de routage

Les DHT offrent une recherche rapide et efficace des données même si le nombre de nœuds qui constituent le réseau est très grand ou la taille des données augmente considérablement. Le routage d'une requête sur DHT demande *O(log n)* sauts dans la plupart des DHT, avec n étant le nombre de nœuds constituant le réseau. Par exemple, dans un réseau à 210 nœuds, le nombre de sauts est de 10 sauts pour une base binaire.

b. Passage à l'échelle

Le passage à l'échelle est assuré grâce au faible diamètre de routage et à la taille constante ou logarithmique des tables de routage. Ceci offre aux DHT un joker pour le passage à l'échelle.

c. Tolérance aux pannes

L'absence de centralisation augmente la robustesse des DHT face aux pannes. Plusieurs approches ont amélioré la tolérance aux pannes DHT par plusieurs méthodes fondées essentiellement sur la réplication des nœuds responsables de la mémorisation des données. Ces méthodes sont souvent efficaces pour éviter l'inaccessibilité et la perte définitive des données. De plus, les DHT permettent une localisation rapide des données ce qui garantit une recherche souvent aboutie.

d. Équilibrage de charge

Le routage sur DHT utilise des fonctions de hachage pour obtenir les identifiants des nœuds et les clés des données. Ceci contribue au partage équilibré des données entre les nœuds constituant le système, il implique un équilibrage de charge entre les nœuds et il évite par la suite la saturation de quelques nœuds par rapport aux autres.

2.2.3 Système publier/souscrire

Plusieurs paradigmes de communication ont été proposés dans le domaine de la programmation répartie. Nous retrouvons le modèle Requête/Réponse puis les modèles Push et Pull. Malgré le progrès réalisé par ces technologies, elles représentent encore des limites sur un réseau à large échelle comme Internet, telles que la scalabilité. Les systèmes publier/souscrire apparaissent donc pour répondre aux défis de la programmation répartie. Parmi les avantages majeurs de ces systèmes est l'expression des besoins des utilisateurs à l'avance, sous forme de requêtes persistantes (dites souscriptions), ce qui permet de répondre à leurs besoins en temps réel. De plus, ces systèmes offrent un découplage fort entre les entités participantes,

ce qui renforce la scalabilité de ces systèmes.

Dans ce qui suit, nous définissons les systèmes publier/souscrire, leurs architectures, leurs langages de souscription et leurs caractéristiques.

2.2.3.1 Définition

Un système publier/souscrire, nommé aussi système basé évènements, est un nouveau modèle de communication entre deux types d'acteurs : producteur et consommateur, à travers un service d'évènements. Les consommateurs expriment leurs intérêts sous forme de souscriptions mémorisées chez le service d'évènements. Les producteurs (fournisseurs) publient des informations sous forme d'évènements au service d'évènements. Ce dernier est formé par des serveurs d'évènements (brokers) distribués, responsables du filtrage des évènements et de leur routage vers les consommateurs sous forme de notifications. Le service d'évènements ne notifie le consommateur que si l'évènement relie l'intérêt (souscription) enregistré [50, 51]. La Figure 2.3 présente le principe de fonctionnement, avec les composants de base, d'un système publier/souscrire.

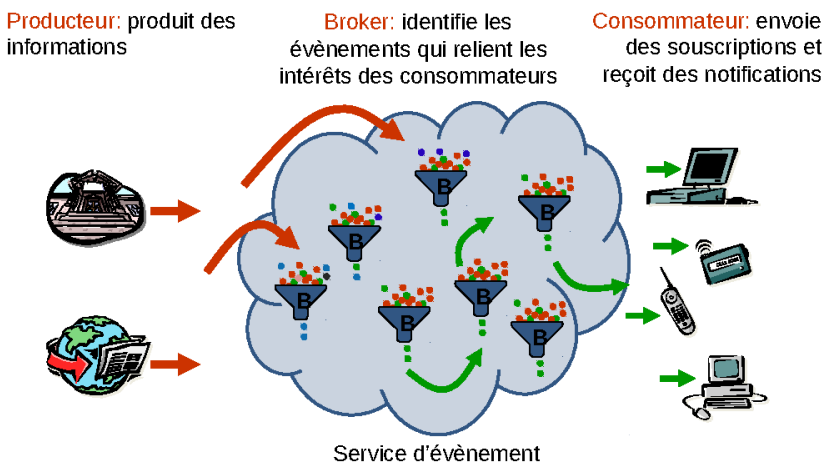


Figure 2.3 – Principe de fonctionnement d'un système publier/souscrire

2.2.3.2 Architecture

Un système basé évènement est un paradigme de communication distribué qui est formé par un ensemble de composants : les serveurs d'évènements qui forment eux-mêmes le service d'évènements et les utilisateurs (consommateurs ou producteurs). L'architecture d'un

système publier/souscrire dépend donc de la relation entre ces composants. Nous retrouvons principalement deux classes d'architectures : le modèle client-serveur et le modèle pair-à-pair.

1. Modèle client-serveur

Suivant ce modèle, un composant peut servir comme un serveur d'évènements ou un utilisateur. Un serveur d'évènements reçoit les évènements, les mémorise et les diffuse. Un serveur d'évènements collabore avec les autres serveurs d'évènements pour atteindre quelques propriétés, telles que la scalabilité. Un utilisateur agit comme producteur, consommateur ou les deux à la fois. Nous retrouvons quatre types de topologies décrivant la connexion entre les serveurs d'évènements [52] :

- topologie en étoile (serveur centralisé) ;
- topologie hiérarchique ;
- topologie en anneau ;
- topologie de polygone irrégulier.

a. Topologie en étoile

Suivant cette topologie, le service d'évènements repose sur une entité unique nommée broker. Cette entité est responsable de mémoriser les souscriptions provenant des consommateurs et de notifier les consommateurs selon leurs souscriptions. Cette topologie permet l'échange d'évènements entre les producteurs et les consommateurs de manière asynchrone. Comme le montre la Figure 2.4, quatre producteurs, de *P1* à *P4*, publient des évènements pour être reçus par les souscrits de *A1* à *A4*. Un souscrit peut recevoir des évènements de l'un des fournisseurs. La notion clé est l'existence d'un seul serveur qui agit en médiateur entre les producteurs et les souscrits (consommateurs) [52].

Cette topologie est assez simple mais elle a l'inconvénient qu'une panne affectant un broker du service d'évènements cause la panne de tout le système. De plus, un système publier/souscrire avec cette topologie souffre du problème de scalabilité puisqu'elle ne permet pas de supporter un grand nombre d'utilisateurs et de gérer un grand nombre d'évènements.

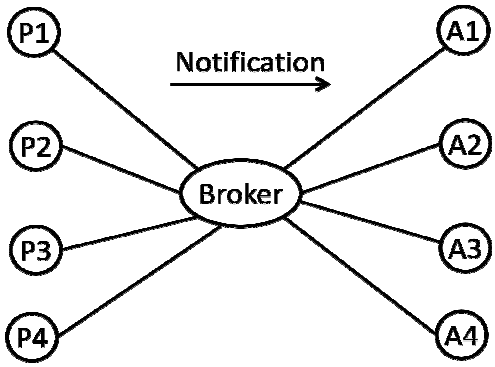


Figure 2.4 – Topologie centralisée d'un système publier/souscrire

b. Topologie hiérarchique

La topologie hiérarchique est distinguée par la relation hiérarchique entre les serveurs d'évènements. Comme le montre la Figure 2.5, chaque serveur, identifié de *B1* à *B5*, sert un certain nombre d'utilisateurs, *C1* à *C6*. Les utilisateurs peuvent être soit des souscrits ou des producteurs. Les serveurs d'évènements se connectent à un serveur parent (root). Dans cette topologie, la communication entre serveur-serveur et client-serveur suit le même protocole. Ainsi, un serveur ne distingue pas entre les autres serveurs et ses clients. Le but de l'organisation hiérarchique est la scalabilité. Le serveur parent reçoit les évènements publiés et les souscriptions émises de l'ensemble de ses utilisateurs, mais transmet à son sous-arbre uniquement les évènements destinés à lui. Le serveur parent agit comme un contrôleur d'accès en maintenant le trafic de son sous-arbre [52].

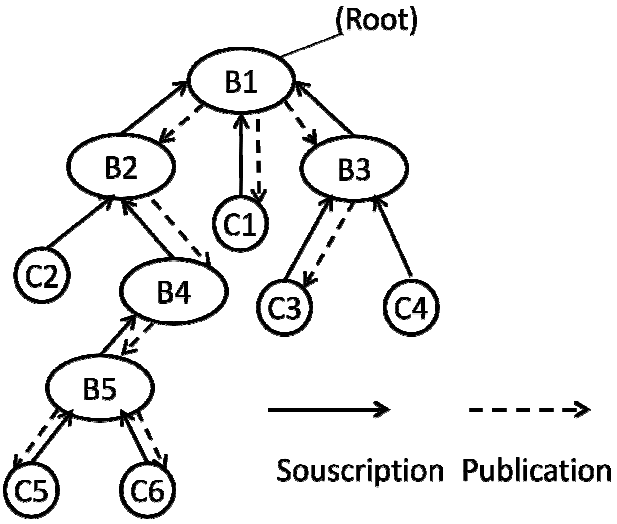


Figure 2.5 – Topologie hiérarchique d'un système publier/souscrire

Nous notons pour cette topologie qu'une panne affectant un serveur de l'arbre cause la panne de tous les serveurs qui lui sont connectés (au dessous de lui dans la hiérarchie).

c. Topologie en anneau

Comme montré par la Figure 2.6, la relation entre les serveurs d'évènements est bidirectionnelle. Le graphe de connexion entre les serveurs d'évènements est un anneau. La communication entre les serveurs se fait selon un protocole de communication bidirectionnelle pour échanger les souscriptions et les publications. Le protocole serveur-serveur est différent du protocole client-serveur dans le type et dans la quantité des informations échangées. Pour une communication serveur-serveur, les deux nœuds d'extrémité maintiennent des informations l'un pour l'autre. Mais pour une connexion client-serveur, le client peut générer une souscription et être aussi le destinataire final d'une publication, et de l'autre côté, le serveur sert comme un point d'accès ou un routeur pour transmettre des messages [52].

Le protocole de routage pour cette topologie est assez compliqué surtout pour détecter les mailles et le plus court chemin pour l'acheminement des évènements entre les serveurs.

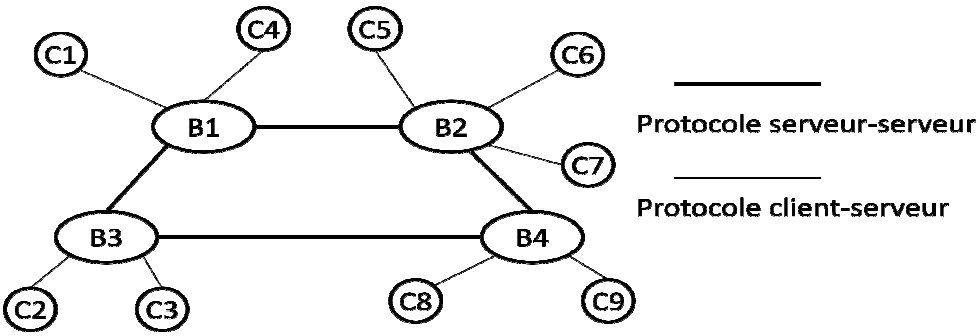


Figure 2.6 – Topologie en anneau d'un système publier/souscrire

d. Topologie de polygone irrégulier

La topologie de polygone irrégulier est une topologie en anneau générique. Ainsi, le graphe du réseau est un graphe générique (voir la Figure 2.7) où il peut exister plusieurs chemins entre deux serveurs. Semblablement à la topologie en anneau, cette topologie permet des communications bidirectionnelles entre deux serveurs.

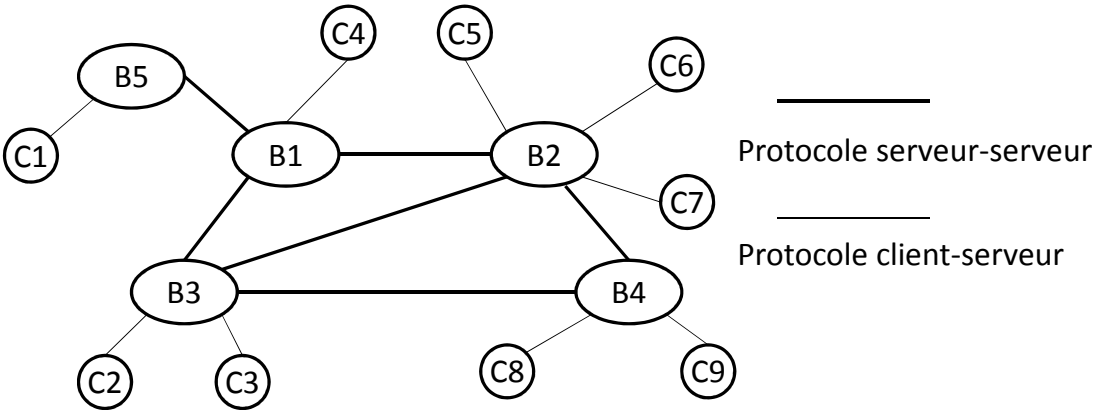


Figure 2.7 – Topologie de polygone irrégulier d'un système publier/souscrire

Comme la topologie en anneau, le protocole de routage sur cette topologie est assez compliqué pour la détection du chemin le plus court vu l'existence d'anneaux.

2. Modèle Pair-à-Pair

Dans le modèle pair-à-pair, tous les nœuds sont égaux comme présenté dans la Figure 2.8. Il n'y a ni nœuds serveurs, ni nœuds clients dans ce modèle. Ainsi, chaque nœud peut agir comme un producteur, un consommateur (souscrit), une racine d'un arbre de multicast, un nœud interne d'un arbre de multicast, ou une combinaison raisonnable de ceux-ci.

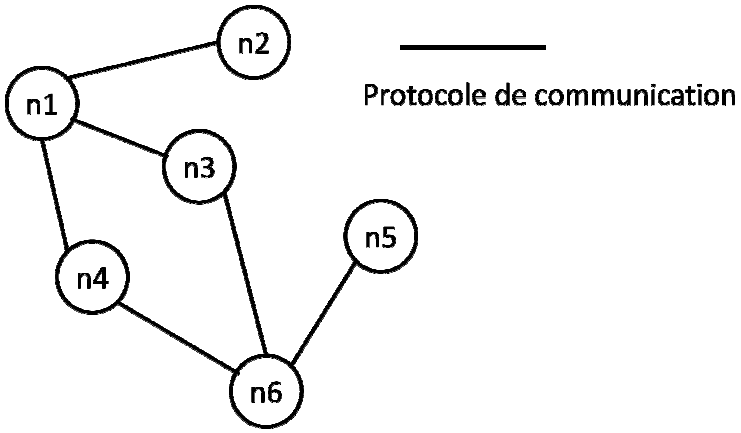


Figure 2.8 – Le modèle pair-à-pair

Cette topologie évite le recours à des serveurs externes pour développer le service d'évènements. En effet, les terminaux des utilisateurs du système publier/souscrire peuvent être utilisés comme des serveurs d'évènements. Ceci minimise le coût de développement d'une plateforme fondée sur un système publier/souscrire de topologie P2P.

2.2.3.3 Filtrage dans les systèmes publier/souscrire

Dans le modèle de publier/souscrire, les souscrits reçoivent seulement un sous-ensemble de l'ensemble des messages publiés. Le processus de sélection des messages au niveau du service d'évènements est appelé filtrage. Il existe deux formes courantes de filtrage : filtrage basé sujet et filtrage basé contenu [53].

1. Filtrage basé sujet

Dans les systèmes basés sujets, les souscrits se joignent à un ou plusieurs groupes à la fois. Chaque groupe partage un sujet d'intérêt [54]. Chaque souscrit énonce ses intérêts sous forme de sujets ou aussi de thèmes. Il peut appartenir donc à un ou plusieurs groupes. Les publications qui appartiennent à un sujet sont diffusées à tous les membres du groupe de ce sujet. Par conséquent, les producteurs et les souscrits doivent spécifier explicitement le groupe auquel ils souhaitent appartenir.

Comme exemples de ce système basé sujet, nous citons Scribe [20] et Bayeux [55].

2. Filtrage basé contenu

Dans les systèmes publier/souscrire basés contenus, le matching des souscriptions et des publications est fondé sur le contenu des évènements. Ces systèmes sont plus souples et utiles

puisque les consommateurs peuvent spécifier leurs intérêts avec plus de précision en utilisant un ensemble de prédicats. La construction de ce type de système nécessite une méthode d'appariement, dit matching, efficace et flexible pour tous les contenus des publications et des souscriptions.
Les exemples de systèmes basés contenus les plus connus sont Siena [53] et Elvin [56].

2.2.3.4 Caractéristiques d'un système publier/souscrire

1. Découplage tridimensionnel

Les systèmes publier/souscrire offrent un découplage tridimensionnel entre les producteurs et consommateurs : découplage dans le temps, découplage dans l'espace et découplage de synchronisation.

a. Découplage dans le temps

Dans le modèle publier/souscrire, les producteurs et les consommateurs n'ont pas besoin d'être connectés en même temps pour produire et consommer des évènements. Les consommateurs peuvent envoyer leurs souscriptions et recevoir des publications même si les producteurs ne sont pas connectés. De même, les producteurs peuvent publier des évènements même si les consommateurs intéressés sont déconnectés. Le matching des souscriptions/publications se fait aussi indépendamment de l'état des utilisateurs. Ce type de découplage est montré par la Figure 2.9.

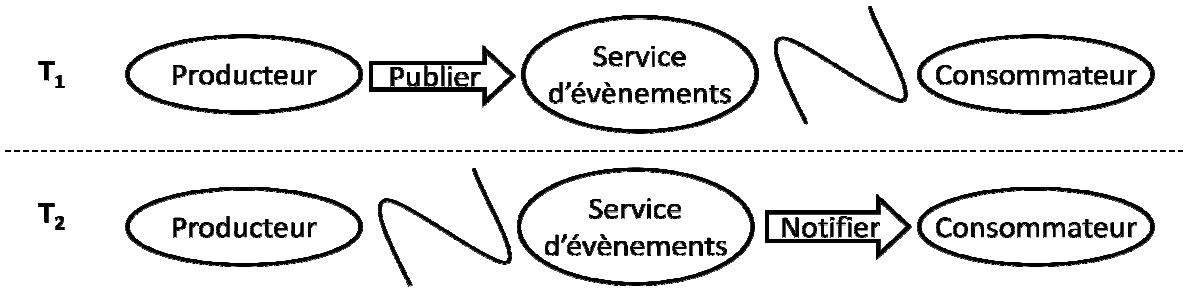


Figure 2.9 – Découplage dans le temps

b. Découplage dans l'espace

Les consommateurs et les producteurs n'ont pas besoin de se reconnaître les uns aux autres du point de vue identifiant, adresse, emplacement ou n'importe quelle autre information. Par conséquent, les producteurs envoient leurs publications au service d'évènements

pour être consommées par les consommateurs indépendamment de leurs états. Ce type de découplage est montré par la Figure 2.10.

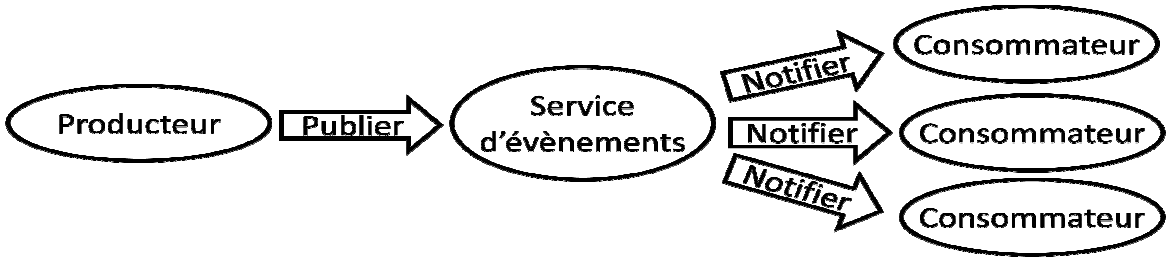


Figure 2.10 – Découplage dans l'espace

c. Découplage de synchronisation

La communication entre utilisateurs est complètement asynchrone. En effet, les consommateurs peuvent être notifiés dans le temps pendant lequel les producteurs publient d'autres évènements (voir Figure 2.11).

Figure 2.11 – Découplage de synchronisation

Ce découplage tridimensionnel permet à un grand nombre de producteurs et de consommateurs une facilité de communication et améliore la mise à l'échelle du système (scalability).

2. Expressivité

Les systèmes publier/souscrire sont utilisés essentiellement pour la distribution sélective des données selon les intérêts des utilisateurs consommateurs. L'expressivité est donc une propriété primordiale pour ces systèmes. Elle dépend du type de filtrage et du langage de souscription utilisé. Un système basé contenu est plus expressif que celui basé sujet. Voire dans les systèmes basés contenus, l'expressivité s'améliore avec ceux qui permettent aux consommateurs d'exprimer leurs intérêts en détails et qui utilisent un processus de matching exact pour répondre au maximum aux besoins des utilisateurs. Ceci dépend aussi du modèle de souscription. Il peut être sous la forme (type, nom, valeur, opérateur) qui peut utiliser

aussi des relations logiques pour satisfaire plusieurs valeurs [53]. Plusieurs autres modèles ont été proposés pour améliorer l'expressivité : SQL, XML, RDF et OWL [52].

2.2.3.5 Système publier/souscrire basé DHT : Scribe

Pour tirer profit des avantages des systèmes publier/souscrire et des réseaux P2P structurés (DHT), nous étudions en particulier les systèmes publier/souscrire d'architecture P2P structurée. Nous avons trouvé dans la littérature que tous ces systèmes sont basés sujets [20, 49, 55, 57] et des travaux de recherches ont visé à améliorer leur expressivité en les rendant basés contenus [58, 59, 60, 61]. Nous présentons ici Scribe [20] puisqu'il est le plus connu et puisque tous les systèmes basés DHT ne différent principalement que par le routage de la DHT utilisée.

Scribe est fondé sur un arbre de multicast construit sur la DHT Pastry. Chaque broker est représenté par un pair de Pastry qui est identifié par une séquence de digits de 128 bits. La Figure 2.12 détaille les étapes de construction de l'arbre de multicast.
Au cours de la phase de souscription, les utilisateurs définissent différents sujets auxquels les autres utilisateurs peuvent s'abonner. Quand un nouveau sujet apparait, un nouvel arbre de multicast se crée et un nouveau broker est sélectionné comme nœud responsable de ce nouveau sujet (appelé *root*). Ce nœud est dit aussi nœud *Rendez-vous* puisque les souscriptions et publications du même sujet se rencontrent à ce nœud. Le root doit avoir l'identifiant le plus proche numériquement de la clé générée de la souscription. En allant du souscrit au nœud root, une nouvelle branche de l'arbre s'ajoute. Si un sujet est arrivé à un broker de la branche, alors la souscription en cours ne continue pas le chemin vers le root. Ceci est illustré par l'exemple de la Figure 2.12 : le nœud 370 est le root de l'arbre de multicast du topic ayant l'ID 371. Le chemin entre le nœud 56B et le nœud 370 forme une branche de l'arbre de multicast. La souscription provenant du nœud BCE est arrêtée au nœud 37F car ce dernier a été un nœud de passage pour cette même souscription (provenant de 56B). Par conséquent, seulement les nœuds BCE et 3A4 sont ajoutés à l'arbre de multicast. Ceci rend le processus de routage de Scribe plus performant en minimisant le temps mis pour le routage et le nombre de sauts effectués.

Au cours de la phase de publication, l'arbre de multicast créé pendant la phase de souscription est utilisé. Le sujet publié est routé vers le nœud de RDV par le protocole de Pastry. Le nœud RDV devient responsable de la diffusion de la publication à tous les souscrits in-

téressés qui sont liés auparavant à l'arbre de multicast du sujet concerné. Cette diffusion se fait donc en suivant les branches de l'arbre de multicast approprié.

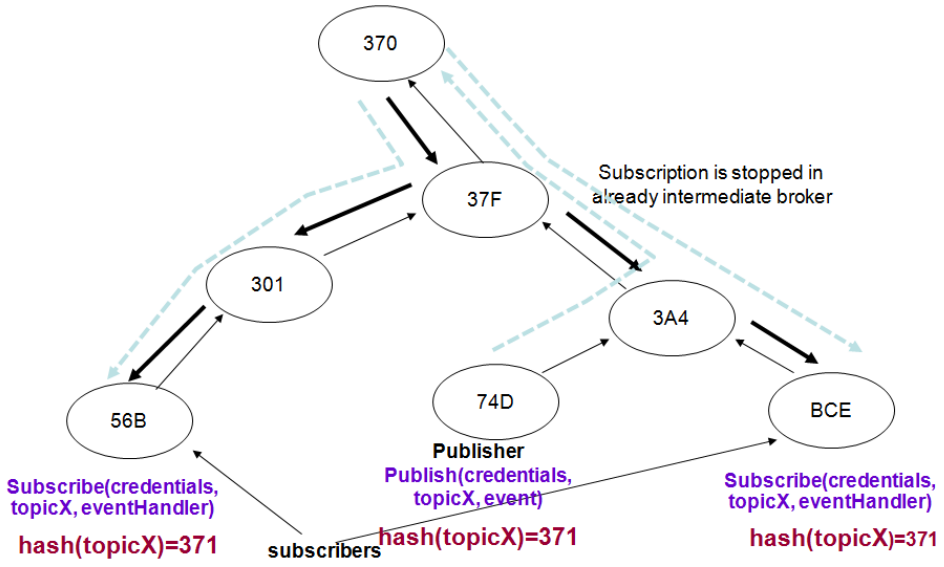


Figure 2.12 – Arbre de multicast de Scribe

2.2.4 Aspect sémantique : ontologie

L'aspect sémantique est très important pour notre plateforme MULUS afin de mieux répondre aux besoins des utilisateurs. MULUS doit être alors capable d'analyser les intérêts exprimés par les utilisateurs et de répondre à ces intérêts selon les contenus publier sur le réseau. Ceci nécessite une représentation et modélisation des connaissances permettant d'identifier les relations sémantiques entre concepts afin d'assurer une analyse sémantique entre les besoins exprimés et les contenus publiés. Nous avons choisi alors les ontologies parce qu'elles sont considérées comme étant le modèle de représentation le plus expressif et le plus formel par rapport aux autres modèles existants (glossaire, thésaurus et taxonomie) [62]. En effet, elles permettent de représenter la sémantique qui existe entre les différents termes d'un domaine ainsi que les relations entre eux.

2.3 Vue d'ensemble de l'approche proposée

Comme étudié dans le premier chapitre, les RS sont de plus en plus développés afin de couvrir les besoins des différents utilisateurs. Mais, ils présentent encore plusieurs lacunes au

niveau de l'architecture, la disponibilité des données, la livraison sans perte et l'expression des intentions des utilisateurs devant la grande quantité de données produites sur Internet. En effet, nous remarquons qu'un échange de données est souvent inutile ou se produit à des moments inadéquats pour l'utilisateur. Cet échange se fait souvent selon des liens d'amitiés (exemple Facebook, LinkedIn, Whatsapp, etc.). D'après notre étude bibliographique, nous avons constaté que la plupart des RS déjà déployé sur Internet utilisent une infrastructure centralisée formée par un nombre limité de serveurs sous le contrôle d'un fournisseur de services. Ce dernier soit exige des frais pour l'utilisation du RS, soit impose des contraintes telles que l'espace de stockage limité, les services offerts limités, les problèmes de sécurité et de confidentialité des données personnelles, etc. Ainsi, la déviation du centralisée à une infrastructure distribuée parait une solution prometteuse du point de vue performance, sécurité, tolérance aux pannes et scalabilité.

Un système publier/souscrire est un paradigme populaire dans lequel les utilisateurs peuvent souscrire ou publier des contenus sur la base de leurs intérêts. Ces deux phénomènes des RS émergents et des systèmes publier/souscrire nous ont motivés à présenter une proposition pour l'amélioration des RS par l'intégration d'un système publier/souscrire afin de les rendre à la fois plus souhaitables et captivant pour les utilisateurs. Après une étude détaillée des systèmes publier/souscrire ainsi que leurs architectures, nous avons choisi d'adopter ce type de systèmes dans l'approche que nous allons proposer dans ce qui suit. Dans cette thèse, nous voulons montrer comment les fonctionnalités d'un système publier/souscrire peuvent être adaptées dans les RS fondés sur l'intérêt de l'utilisateur dans l'abonnement et la publication.

Nous expliquons dans cette section l'architecture de base que nous proposons ainsi qu'une décomposition de la problématique de cette thèse.

2.3.1 Présentation générale de la plateforme MULUS

Dans le premier chapitre, nous avons identifié principalement les défis à résoudre dans cette thèse : (i) le partage de contenus composites selon les intentions des utilisateurs (ii) la décentralisation des RS pour plus de sécurité et de scalabilité (iii) assurer la disponibilité de données ainsi que leur livraison sans perte.

Une solution qui peut faire face à ces défis est l'utilisation des systèmes publier/souscrire. Notre approche consiste à concevoir et implémenter un RSD fondé sur un système

publier/souscrire sémantique, de topologie P2P structurée et supportant la composition d'évènements. Ce RSD, que nous avons appelé **"MULUS"**, assure le partage de données, en particulier des contenus **MUL**timédias composites entre les cl**US**ters des utilisateurs.

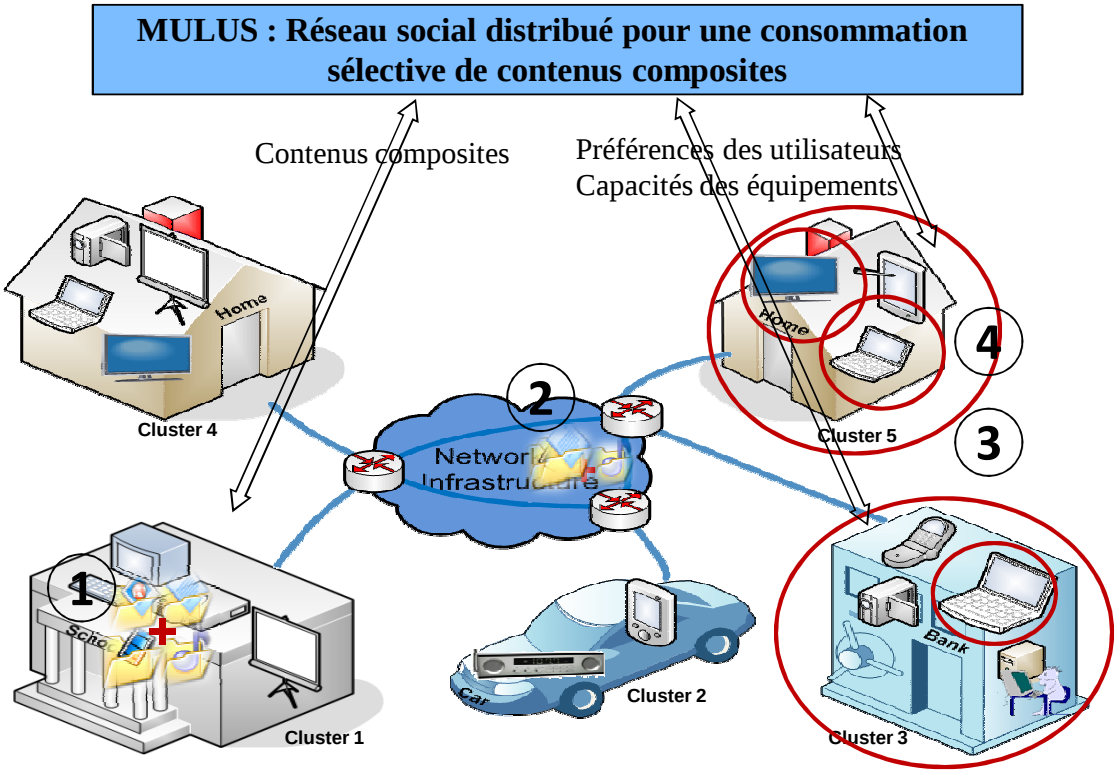


Figure 2.13 – Interaction avec MULUS pour le partage de données composites

La Figure 2.13 présente l'interaction avec la plateforme MULUS et les étapes primordiales pour la livraison souhaitée [11]. Après qu'un utilisateur exprime son besoin composite et lorsqu'un producteur fournit un ensemble de données (1), ces éléments seront décomposés au niveau du réseau intermédiaire. Ensuite, une composition d'éléments (2) et une sélection de clusters (3) se font selon les intérêts des utilisateurs pour que la composition soit acheminée uniquement vers les souscrits. À l'arrivée au cluster d'un utilisateur, la sélection des équipements se fait selon la description des capacités et les préférences des utilisateurs (4). Nous finissons par la distribution des éléments sur les équipements sélectionnés au niveau du cluster.

2.3.2 Architecture de MULUS

Pour gérer la distribution correcte de données composites entre les utilisateurs, MULUS doit assurer le filtrage des données partagées sur le réseau. Pour ce faire, MULUS repose sur un système publier/souscrire de topologie P2P structurée qui communique les clusters des différents utilisateurs à travers une entité *Home Box Entity* (HBE) (voir Figure 2.14). Cette entité est déployée sur un dispositif sélectionné de chaque cluster jouant le rôle d'une gateway. Une fois le nouveau paquet délivré aux consommateurs des clusters correspondants, nous avons besoin de sélectionner les appareils du cluster capables de supporter les éléments reçus. Cette entité joue le rôle d'un serveur d'évènements une deuxième fois pour connecter les différents équipements au niveau local du cluster. L'architecture de cette entité sera détaillée dans le Chapitre 3.

La mise à l'échelle de notre plateforme est garantie grâce à la topologie P2P structurée du système publier/souscrire reliant les différents clusters. Cette topologie utilise la DHT qui se caractérise aussi par l'auto-organisation des nœuds et l'équilibrage de charge (le load-balancing) entre les nœuds qui composent le service d'évènements du système publier/souscrire. La DHT est connue aussi par le routage flexible entre les nœuds fondé sur le routage des pairs (clé, valeur) et par la scalabilité. Avec ce protocole, les DHT procurent aux développeurs un haut niveau d'abstraction pour implémenter un système de stockage persistant et à large échelle. Ils masquent la complexité de routage, de tolérance aux pannes et de réplication de données. Par conséquent, ils sont de plus en plus utilisés pour développer des applications dont la fiabilité est primordiale, telles que la gestion répartie de fichiers, les systèmes de sauvegarde, ou les applications de distribution de données. Pour notre plateforme MULUS, nous choisissons le système Scribe que nous avons présenté dans la section 2.2.3.5. Pour la diversité des intérêts des utilisateurs et l'hétérogénéité des données, notre plateforme a recours à la description sémantique pour permettre aux utilisateurs une meilleure précision lors de la description de leurs intentions. Vu la diversité de la description sémantique, nous utilisons une ontologie de domaine unifiée entre tous les utilisateurs de notre plateforme pour l'étendre à des individus exprimant leurs différents besoins. Nous expliquerons plus tard notre approche pour l'utilisation de l'ontologie sur le système Scribe.

Comme le montre la Figure 2.14, les serveurs d'évènements de notre système publier/-souscrire sont des équipements choisis des différents clusters. Ce choix est justifié par les avantages identifiés dans la section précédente du modèle P2P de ces systèmes. Néanmoins,

ces équipements sont réellement trop dynamiques sur le réseau vu l'utilisation fréquente des équipements mobiles qui favorise la connexion/déconnexion répétée sur le réseau. En conséquence, nous devons résoudre le problème de dynamisme fort des serveurs d'évènements d'un système publier/souscrire P2P basé DHT. Nous traiterons ce problème dans le Chapitre 5.

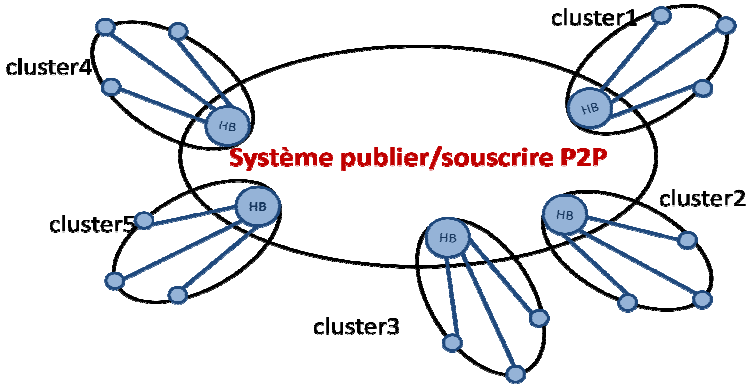


Figure 2.14 – Architecture de la plateforme MULUS [11]

2.4 Caractéristiques de MULUS

Nous présentons dans cette section les caractéristiques de notre plateforme MULUS.

2.4.1 Scalabilité

D'après une étude bibliographique, les RS sont les plus utilisés pour le partage de contenus entre des utilisateurs distants. Les RS actuels sont centralisés où chaque RS est contrôlé par une seule entité. Comme les RS sont utilisés par une grande population, l'exigence en termes de sécurité et de scalabilité est très marquante car une panne ou une attaque du système engendre un mal-fonctionnement voire un dysfonctionnement total.

Par ailleurs, nous avons constaté que les réseaux distribués permettent de supporter un grand nombre d'utilisateurs et une grande quantité de données. En étudiant les différentes architectures des systèmes distribués, nous constatons que les systèmes P2P structurés sont les plus efficaces du point de vue sécurité, scalabilité, auto-reconfiguration et tolérance aux pannes. De plus, grâce à ce type de système, nous n'avons plus besoin de serveurs à installer ou à connecter pour servir comme intermédiaires de routage ou de stockage de données. De plus, nous aurons une augmentation proportionnelle entre le nombre d'utilisateurs avec leurs

équipements et la quantité de données partagées sur le RS.
En conséquence, nous avons choisi d'adopter l'architecture P2P structurée pour le développement de notre plateforme.

2.4.2 Consommation sélective des contenus partagés

Dans les RS actuels, une publication peut être acheminée à tout le monde dans le système ou uniquement aux amis associés au producteur selon le type du RS. Ces RS ne tiennent pas compte de l'intérêt des personnes qui reçoivent cette annonce. En effet, le RS ne fait pas la correspondance entre l'intérêt des utilisateurs consommateurs et les publications reçues de la part des producteurs. Pour pouvoir tenir compte des besoins des utilisateurs, nous explorons l'idée de l'intégration d'un système publier/souscrire fondé sur le matching entre les intérêts des souscrits et les publications. D'après l'étude des systèmes publier/souscrire, un système basé contenu s'est avérée une solution efficace pour la consommation sélective des contenus partagés. Il permet aux consommateurs d'exprimer leurs intérêts en détails sous forme d'une souscription à enregistrer chez le service d'évènements pour que ce dernier puisse notifier les souscrits intéressés lorsqu'un évènement provient d'un producteur devient disponible. Avec cela, plus le système offre un langage de souscription riche, plus l'utilisateur devient satisfait du service. Nous devons trouver un langage de souscription très descriptif qui permet une meilleure diffusion de l'information ; d'autant plus que les données partagées sont en général peu satisfaisantes, font épuiser la bande passante et font supporter l'utilisateur un surcoût de son débit. Avec l'ontologie, la description des intérêts devient plus souple et utile puisque les souscrits peuvent spécifier leurs intérêts avec plus de précision en utilisant un ensemble de prédicats. La principale raison de l'utilisation de l'ontologie est de fournir un vocabulaire riche, bien structuré, couvrant un domaine donné (comme le domaine de multimédia) et commun pour tous les utilisateurs. La meilleure solution est alors d'adapter un système publier/souscrire sémantique utilisant l'ontologie pour permettre aux utilisateurs de mieux exprimer leurs besoins.

D'une part, nous visons à utiliser un système publier/souscrire basé contenu et distribué pour répondre aux besoins d'un grand nombre d'utilisateurs géographiquement éloignés. D'autre part, le service d'évènements doit être de topologie P2P structurée pour surmonter les points faibles des systèmes existants. Ainsi, l'utilisation de la sémantique avec la topologie P2P structurée constitue un enjeu et un défi scientifique important. En effet, le routage

sur ce type de topologie ne permet que de vérifier l'égalité entre les clés de routage alors que la description sémantique permet l'utilisation des mots syntaxiquement différents mais sémantiquement équivalents. Nous traiterons alors ce problème dans le Chapitre 3.

2.4.3 Composition des contenus partagés

Dans le cadre de l'expressivité, nous notons aussi qu'une souscription précise doit permettre d'organiser la réception des publications, en précisant des contraintes logiques et temporelles. En effet, l'accélération du rythme de vie demande une bonne organisation des activités professionnelles et de loisir. Ceci donne naissance à une composition et agrégation de données définie par la notion de bundle. Cette composition peut être définie par des relations logiques et temporelles entre ces éléments selon les intérêts exprimés par les utilisateurs. Les éléments composants un bundle, lors de sa livraison, peuvent être fournis par plusieurs utilisateurs. Alors leurs agrégations selon des relations logiques et temporelles demeurent une tâche difficile surtout pour un système temps réel distribué. La cohérence de regroupement des évènements demande un effort de coordination entre les serveurs d'évènements. Si les évènements sont mal regroupés, les utilisateurs seront submergés par des publications inutiles.

Les systèmes publier/souscrire traitent aussi ce problème avec la notion d'évènement composite (*composite event*) qui permet d'exprimer un intérêt composite de plusieurs autres intérêts reliés par des relations logiques, temporelles ou spatiales. Les approches trouvées dans la littérature traitent la composition d'évènements dans les systèmes centralisés où le service d'évènements est formé d'un seul broker. Ainsi, le problème de coordination ne se pose pas. D'autres approches sont proposées pour les systèmes distribués mais ne permettent pas de couvrir toutes les contraintes logiques et temporelles. Ces solutions ne sont pas toutes adaptées pour un système publier/souscrire basé DHT. Nous traiterons ce problème avec plus de détails dans le Chapitre 4.

2.4.4 Traitement du dynamisme fort des utilisateurs

Nous rappelons que pour le modèle P2P d'un système publier/souscrire, tous les nœuds (brokers et utilisateurs) sont égaux. Chaque nœud peut agir comme un producteur, souscrit, racine d'un arbre de multicast, un nœud interne d'un arbre de multicast, ou une combinaison

raisonnable de tous ceux-ci. L'avantage majeur de ce modèle est qu'il évite le recours à des serveurs externes pour construire le service d'évènements. En réalité, les terminaux des utilisateurs du système publier/souscrire peuvent être utilisés comme des serveurs d'évènements. Ceci minimise le coût de développement d'une plateforme fondée sur un système publier/-souscrire de modèle P2P. Nous proposons donc d'adapter ce modèle dans notre plateforme.

De nos jours, les nouvelles technologies utilisées favorisent la mobilité des utilisateurs ; d'où résulte un dynamisme fort des nœuds composants le système publier/souscrire vu les connexions et déconnexions fréquentes et avec des durées de session trop variées. Ainsi, nous devons bien considérer le dynamisme fort des utilisateurs lors du partage des données, vu que le partage se fait directement entre eux, sans passer par un serveur intermédiaire externe. Nous devons permettre aux utilisateurs de recevoir les contenus partagés même s'ils étaient déconnectés au moment du partage et ceci se fait toujours selon les intérêts exprimés auparavant. De plus, l'acheminement entre le producteur et le ou les utilisateurs consommateurs doit être toujours assuré en passant par d'autres utilisateurs intermédiaires dont leurs états peuvent changer (connecté/déconnecté). Nous devons trouver non seulement les chemins disponibles mais aussi ceux qui satisfont les contraintes de QdS souhaitées. Puisque nous adoptons la topologie P2P structurées, nous retrouvons dans la littérature beaucoup de travaux qui explorent le dynamisme fort des nœuds sur DHT mais qui ne sont pas assez efficaces pour un système publier/souscrire basé DHT. En effet, tous ces travaux proposent des méthodes de réplication de données sur les nœuds de DHT pour pouvoir les retrouver en utilisant simplement le routage de DHT. Sur un système publier/souscrire basé DHT, ceci devient insuffisant puisque ces systèmes utilisent tout un arbre de multicast (mémorisation du chemin du souscrits jusqu'au nœud responsable du sujet) pour pouvoir acheminer les publications du producteur au consommateur en passant par les nœuds intermédiaires. Donc, la réplication du nœud responsable est insuffisante si les nœuds intermédiaires sont aussi dynamiques. Nous aborderons ce problème dans le Chapitre 5.

2.5 Conclusion

Dans la première partie de ce chapitre, nous avons présenté les concepts de base qui nous aide à atteindre les objectifs fixés dans le premier chapitre. Nous avons commencé par présenter les systèmes distribués avec les différentes topologies en identifiant les caractéristiques

de chacune. Ensuite, nous avons étudié les systèmes publier/souscrire comme étant un paradigme de communication permettant une distribution sélective du contenu et un découplage tridimensionnel. Ceci nous a permis de faire le choix pour développer un RSD performant du point de vue scalabilité, temps de réponse et trafic sur le réseau. Enfin, nous avons présenté l'architecture de notre plateforme MULUS et nous avons décomposé notre problématique pour pouvoir la traiter dans les chapitres suivants.

Chapitre

3 Traitement de la sémantique dans MULUS

3.1 Introduction

Les RS sont de plus en plus développés pour couvrir les demandes des utilisateurs. Mais comme nous l'avons signalé dans le premier chapitre, ils présentent encore plusieurs lacunes surtout au niveau de l'expression des intérêts des utilisateurs devant la grande quantité de données partagées et produites sur Internet. En effet, nous remarquons qu'un échange de données est souvent inutile pour l'utilisateur avec la majorité des RS tels que Facebook. Pour d'autres RS tels que YouTube, les requêtes des utilisateurs ne sont pas persistantes et efficaces pour exprimer les besoins exacts des utilisateurs. Pour remédier à ces lacunes, nous avons proposé l'intégration d'un système publier/souscrire basé contenu assurant une distribution sélective de contenus. Plus le système, tel que le notre, offre un langage de souscription riche et sémantique, plus l'utilisateur devient satisfait du service.

Comme nous avons choisi d'utiliser Scribe, nous nous trouvons face au problème de routage sémantique sur DHT. En effet, le routage sur ce type de topologie ne permet que de vérifier l'égalité entre les clés de routage, tandis que la description sémantique permet l'utilisation des mots syntaxiquement différents mais sémantiquement équivalents.

Ce chapitre aborde le deuxième problème de cette thèse qui est le traitement de la sémantique dans notre plateforme MULUS pour répondre aux besoins des utilisateurs. Nous commençons par une étude bibliographique pour étudier la sémantique dans les réseaux publier/souscrire fondés sur DHT. Ensuite, nous définissons les concepts de base de l'ontologie et nous détaillons notre approche pour le traitement de la sémantique dans MULUS. Enfin, pour valider notre approche, nous décrivons les expérimentations faites pour montrer que l'intégration d'un système publier/souscrire sémantique réduit le trafic sur le réseau.

3.2 État de l'art : traitement de la sémantique

Le sujet de la sémantique, à notre connaissance, n'a pas été étudié jusqu'à présent sur les RS ou les RSD. Nous recherchons alors ce sujet sur les systèmes publier/souscrire que nous avons choisis comme couche de base de notre plateforme. Nous commençons par les systèmes publier/souscrire d'architecture P2P structurée. Ensuite, nous étudions la sémantique sur les systèmes publier/souscrire centralisés et hiérarchiques.

3.2.1 Traitement de l'expressivité dans les systèmes publier/souscrire basés DHT

Quelques travaux ont mené une recherche pour améliorer l'expressivité sur les systèmes publier/souscrire basé DHT comme Scribe [20], P2P-ToPSS [59], Willow [60], PastryStrings [58] et Meghdoot [61].

Comme nous l'avons expliqué dans la section 2.2.3.5 du Chapitre 2, Scribe est fondé sur la DHT Pastry pour construire son arbre de Multicast pour chaque topic. Il est scalable et il supporte la tolérance aux pannes mais sa limite majeure est l'expressivité. En effet, il est basé sujet avec des souscriptions de la forme "topic=x".

P2P-ToPSS étend Scribe pour améliorer l'expressivité du système. Il améliore la forme de la souscription seulement pour le type numérique scalaire. Il définit la notion de "schema" pour fixer les valeurs numériques par intervalle. Ensuite, si un utilisateur souhaite s'inscrire pour avoir une valeur appartenant à un intervalle de valeurs, P2P-ToPSS utilise le *schema* défini pour diviser cette souscription en un ensemble de valeurs dont chacune présente un sujet d'une souscription simple. Cette approche présente des inconvénients majeurs. Le premier inconvénient est que P2P-ToPSS surcharge sa DHT Pastry par la gestion d'un grand nombre d'arbres de Multicast même pour une souscription d'un seul utilisateur. Le deuxième inconvénient est que ce système n'est pas efficace pour une souscription sur un intervalle ouvert de valeurs ou sur un intervalle de valeurs réelles. De plus, il n'est pas valable pour des souscriptions sur des données autres que le numérique (entier). Ainsi, P2P-ToPSS souffre d'un problème de scalabilité et d'expressivité en particulier.

PastryStrings traite les souscriptions de type chaîne de caractères et de type numérique seulement avec des opérateurs de comparaison simples. Ce système est scalable mais il souffre aussi du problème d'expression des besoins.

Meghdoot est aussi un système basé contenu sous la DHT CAN [63]. La souscription, sous Meghdoot, est caractérisée par les attributs d'un schéma défini par S=A_1, A_2, ..., A_n où chaque attribut A_i est spécifié par un nom, un type et un domaine sous la forme d'un ensemble *name*, *type*, *min*, *max*.
De même, une souscription sémantique n'est pas assurée par Meghdoot.

3.2.2 Traitement de la sémantique dans les systèmes publier/souscrire centralisés et hiérarchiques

Pour la deuxième classe d'approches, nous trouvons le système Siena proposé par A. Carzaniga *et al.* [50]. C'est un système publier/souscrire basé contenu avec une topologie hiérarchique de son service d'évènements. Il améliore l'expressivité avec une variété de types de souscriptions comme entier, réel et chaîne de caractères, et une variété d'opérateurs de comparaison. Néanmoins, il présente quelques limites telles que le problème d'équilibrage de charge entre les serveurs d'évènements et par conséquent la scalabilité. De plus, la panne d'un serveur père engendre la panne du système.

Keeney *et al.* [64] ont étendu Siena par l'utilisation d'une ontologie pour la description des souscriptions et des publications. Ils ont intégré des relations de couverture (*covering*) avec quelques opérateurs entre les souscriptions envoyées aux serveurs d'évènements. Ceci réduit le coût en particulier le temps mis pour le matching des souscriptions et sa complexité. Cependant, le problème de scalabilité et d'équilibrage de charge entre les serveurs d'évènements persiste encore.

Le dernier travail est celui de Skovronski *et al.* [65]. Ils ont aussi utilisé une ontologie pour décrire les publications et des requêtes SPARQL pour décrire les souscriptions. Ils ont proposé une classification des différentes données partagées par domaines où chaque domaine représente un sujet. Ils ont défini une ontologie de domaine pour chaque sujet. Cette approche a utilisé un service d'évènements centralisé qui maintient toutes les ontologies de domaines des différents sujets. L'avantage principal de cette approche est la sémantique offerte par l'ontologie qui enrichit la description des besoins et rend le matching plus exact. Cependant, cette approche centralisée souffre du problème de scalabilité.

3.2.3 Synthèse

Le Tableau 3.1 résume les principales caractéristiques des différentes approches proposées pour améliorer l'expressivité des systèmes publier/souscrire. Nous remarquons qu'il y a essentiellement trois types d'approches : le premier type assure la scalabilité grâce à la topologie P2P structurée mais il n'améliore pas l'expressivité. Le deuxième type d'approches essaye d'équilibrer entre la scalabilité et l'expressivité. Il améliore peu l'expressivité mais il n'assure pas convenablement la scalabilité vu la topologie hiérarchique de son service d'évènements. Le troisième type d'approches améliore l'expressivité et il utilise l'ontologie pour assurer plus de sémantique. Cependant, il n'est pas scalable à cause de sa topologie centralisée.

Tableau 3.1 – Sémantique des systèmes publier/souscrire

Approche	**Topologie**	**Scalabilité**	**Filtrage**	**Alg. de matching**	**Expressivité**
Scribe	DHT	✓	Sujet	Exact	
P2P-ToPSS	DHT	✓	Sujet	Exact	
Pastry Strings	DHT	✓	Contenu	Exact	
Meghdoot	DHT	✓	Contenu	Exact	
Siena	Hiérarchique		Contenu	Exact	
J. Keeney *et al.*	Hiérarchique		Ontologie	Sémantique	✓
J. Skovronski *et al.*	Centralisée		Ontologie	Sémantique	✓

Nous en concluons que l'utilisation de la sémantique dans un système publier/souscrire basé DHT est un défi réel. D'une part, la sémantique permet d'utiliser des expressions sémantiquement équivalentes mais syntaxiquement différentes. D'autre part, le routage avec DHT est fondé sur les clés calculées par le hachage des données à partager. Ainsi, la différence syntaxique possible avec la sémantique perturbe le routage des messages sur DHT. Dans la suite, nous définissons les concepts de base sur l'ontologie. Puis, nous proposons de faire un mariage entre la sémantique (ontologie) et la DHT pour assurer la sémantique et la scalabilité concurremment.

3.3 Concepts de base

Nous remarquons que l'expressivité des systèmes publier/souscrire est très importante pour pouvoir répondre aux exigences des utilisateurs. Nous devons offrir alors aux utilisateurs la possibilité d'exprimer leurs besoins. Dans le cas d'une large communauté, nous nous retrouvons face à un vocabulaire très varié dont le matching devient plus difficile. En effet, les utilisateurs peuvent exprimer leurs souscriptions ou publications différemment pour le même sujet. D'où le besoin de trouver une façon, pouvant couvrir les notions d'un domaine spécifique, à travers laquelle les machines peuvent analyser des informations sémantiquement. Ceci a donné naissance aux ontologies qui permettent une modélisation expressive et sémantique des informations. Elles sont d'une grande utilité pour la représentation, l'analyse et le traitement sémantique automatisés des intérêts des utilisateurs.

3.3.1 Définition et objectifs d'une ontologie

Dans le domaine de l'informatique, une ontologie est un ensemble de termes et de relations de base décrivant le vocabulaire d'un domaine, ainsi que les règles qui les relient pour pouvoir étendre le vocabulaire [66].

Une ontologie est définie aussi comme la conceptualisation d'un ensemble de connaissances structurées hiérarchiquement pour décrire un domaine particulier avec leurs propriétés et leurs relations pour permettre leur compréhension et leur analyse.

Nous distinguons deux objectifs essentiels de l'ontologie selon [67] :

- le premier objectif est descriptif. En effet, elle permet de représenter clairement l'ensemble des connaissances et des informations décrivant un domaine bien déterminé.
- le deuxième objectif est le raisonnement. Il consiste à réaliser des inférences et à générer automatiquement de nouvelles connaissances grâce à des règles exprimées dans l'ontologie.

3.3.2 Composition d'une ontologie

Pour définir une ontologie, nous devons identifier les concepts d'un domaine de savoir avec les relations entre eux pour représenter la sémantique indépendamment de l'application dans laquelle elle pourra être utilisée. Les composants de base d'une ontologie sont :

- **les concepts (les classes) :** comme dans la programmation orientée objet, une classe représente un ensemble de ressources partageant les mêmes propriétés. Elle peut représenter un objet concret, une action, une notion, une idée, etc.
- **les individus :** de même, à chaque classe est associé un ensemble d'instances appelées des individus. Les individus peuvent être définis aussi comme des exemples de classes. Ils héritent donc toutes propriétés de cette classe.
- **les propriétés :** elles aident à la caractérisation des classes. Pour assurer la sémantique demandée, les concepts doivent être reliés entre eux. Pour cela, chaque concept est caractérisé par un ensemble de propriétés et lié avec les concepts voisins par des relations.
- **les relations :** elles désignent les différentes corrélations et interactions entre les concepts. Ces relations décrivent les associations suivantes : sous-classe de (généralisation ou spécification), partie de (composition ou agrégation), instance de, associé à, est un, etc.

3.3.3 Langage de modélisation d'une ontologie

Les ontologies peuvent être modélisées avec différentes techniques et peuvent être mises en œuvre avec différents types de langages [68]. L'ontologie peut être exprimée de manière informelle, semi-formelle ou rigoureusement formelle selon la taxonomie donnée du domaine. Un langage qui est couramment utilisé pour concevoir l'ontologie est OWL (Web Ontology Language) [68].

OWL est un langage standardisé par l'organisme W3C [1]. Ce langage est inspiré de deux mécanismes de raisonnement DAML [2] et OIL [3]. Il est considéré comme une extension des vocabulaires RDF [4] et RDF Schema représentés par le langage XML [5], pour construire des vocabulaires de création des ontologies.

Un des avantages clés d'OWL est sa capacité à représenter les concepts d'une façon assez intéressante et complexe que le RDF et le RDF Schema. OWL permet d'ajouter plus de formalisation dans l'expression des informations afin de vérifier leur validité. Ceci est

1. World Wide Web Consortium http ://www.w3.org/TR/owl-ref/
2. Darpa Agent Markup Language
3. Ontology Inference Layer
4. Resource Description Framework
5. eXtensible Markup Language

assuré par la définition des restrictions de cardinalités (au moins, au plus et exactement), des disjonctions de classes (par exemple homme et femme), les transitivités (par exemple les propriétés "est égal à", "plus petit que" et "plus grand que" sont transitives), etc.. Ainsi, OWL offre aux machines une capacité sémantique plus importante que RDF et RDF Schema. Il est capable aussi de concevoir plusieurs ontologies dans une seule ontologie en utilisant le mécanisme d'importation *Import*. Ceci accélère le processus de construction des ontologies et permet de réutiliser des ontologies déjà existantes pour les étendre et définir une nouvelle ontologie.

3.3.4 Langage SWRL

Malgré les caractéristiques du langage OWL, il n'est pas un langage efficace en soi. En effet, il ne permet pas de déduire l'enchaînement logique d'une ontologie (c'est-à-dire les faits qui ne sont pas explicitement présents dans l'ontologie mais peuvent être sémantiquement déduits). Pour dépasser cette insuffisance, nous retrouvons dans la littérature le langage SWRL [6]. C'est un langage standardisé par W3C [7] qui permet aux utilisateurs de rédiger des règles en termes de concepts OWL pour fournir des capacités de raisonnement déductif. Ce langage enrichit la sémantique d'une ontologie définie en OWL en utilisant des mécanismes de raisonnement sur les règles [69]. Ces mécanismes sont implémentés dans des moteurs d'inférence. Ces moteurs s'occupent de l'exécution des règles SWRL à travers des ponts connus par le nom "bridge" afin d'automatiser le processus de déduction sur les règles.

3.4 Traitement de la sémantique dans MULUS

L'utilisateur, dans son réseau domestique, peut avoir plusieurs équipements pour la réception des contenus souhaités. Chaque équipement peut supporter des contenus selon ses capacités. Pour bien répondre aux besoins des utilisateurs, il est important alors de leur permettre d'exprimer leurs besoins du point de vue contenus souhaités et équipements à utiliser pour la réception de ces contenus. La sélection des équipements se fait selon les préférences des utilisateurs et les capacités des équipements.
Nous nous proposons de mettre en place une entité nommée *Home Box Entity* (HBE) dans

6. Semantic Web Rule Language
7. World Wide Web Consortium

chaque cluster d'utilisateurs. Cette entité représente les pairs de notre RSD. Elle est responsable d'assurer la sémantique des descriptions des besoins pour permettre à l'utilisateur d'exprimer ses intentions en détails. Elle gère alors une ontologie de domaine que nous avons définie pour mieux répondre aux besoins des utilisateurs. Elle permet aussi de sélectionner les équipements selon leurs capacités et selon les préférences de l'utilisateur.

Notre approche suit une démarche en trois étapes. La première étape [70] consiste à définir, ou utiliser s'il existe, une ontologie de domaine qui couvre une taxonomie large d'un domaine donné. Par exemple, si nous nous intéressons au domaine multimédia (qui couvre déjà les contenus échangés par la majorité des RS), nous pouvons définir notre ontologie de domaine à partir d'une ontologie de domaine définie dans la littérature telle que Multimedia Web Ontology [71]. La taxonomie de notre ontologie de domaine doit être assez structurée pour couvrir le domaine d'application tout en vérifiant les relations d'équivalence sémantique. Ceci nous permet d'encoder les classes et les concepts de cette nouvelle ontologie de domaine pour les utiliser dans le routage et la localisation des nœuds sur DHT.
Une fois l'ontologie du domaine structurée et indexée (écrite en OWL), elle est partagée entre tous les HBE sous la forme de fichiers OWL. La deuxième étape consiste à définir des règles avec SWRL (règles sémantiques) permettant de retrouver les sujets des classes de l'ontologie du domaine présentée en OWL. Ces règles définissent les propriétés nécessaires à vérifier pour juger l'appartenance d'un évènement à un sujet bien déterminé. Ensuite, les utilisateurs peuvent définir un individu de cette ontologie pour décrire les intérêts ou les éléments à partager sur le réseau. Après avoir récupéré la description des intérêts, nous proposons d'utiliser un raisonneur tel que Pellet [8], permettant de retrouver le sujet de la description sémantique en utilisant les règles SWRL et l'ontologie de domaine.
Dans la troisième étape [72], les sujets retrouvés par le raisonneur sont gérés par le HBE pour le routage des contenus sémantiques des évènements (individus de l'ontologie) sur DHT. Ils sont utilisés comme clés de routage des souscriptions/publications. Par ailleurs, dans la phase de publication, nous devons trouver des sujets plus génériques pour notifier les souscrits correspondants. A ce niveau, nous proposons une méthode d'indexation pour remplacer les fonctions de hachage utilisées par DHT et pour retrouver les identifiants des sujets parents sans faire recours à la sémantique. Dans la section suivante, nous décrivons notre méthode d'indexation.

8. https ://github.com/complexible/pellet

Ces étapes sont montrées par la Figure 3.1.

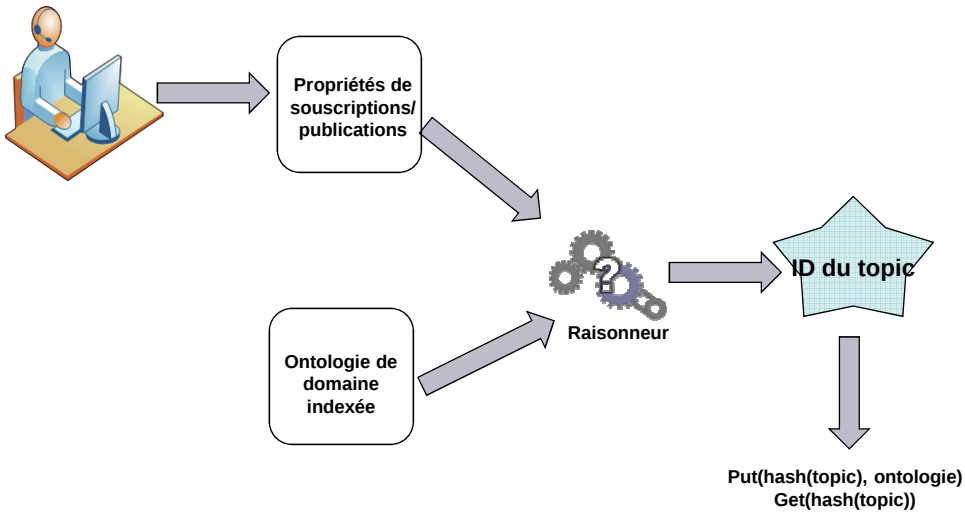


Figure 3.1 – Étapes de souscriptions/publications sur MULUS

Dans la suite, nous détaillons l'architecture du HBE. Puis, nous définissons la structure de l'ontologie de domaine ainsi que le codage sémantique proposé. Nous finissons par la communication entre les HBE distribuées en P2P.

3.4.1 Architecture du *Home Box Entity* (HBE)

Les HBE sont déployées dans chaque réseau domicile ou cluster pour assurer les fonctionnalités sollicitées pour la gestion interne des clusters et le partage des contenus avec les autres serveurs d'évènements. Chaque HBE permet à l'utilisateur d'exprimer en détails son besoin. La Figure 3.2 montre la composition de cette entité.
Dans la suite, nous détaillons la composition de chaque HBE. La communication entre les HBE sera expliquée dans la section 3.4.3.

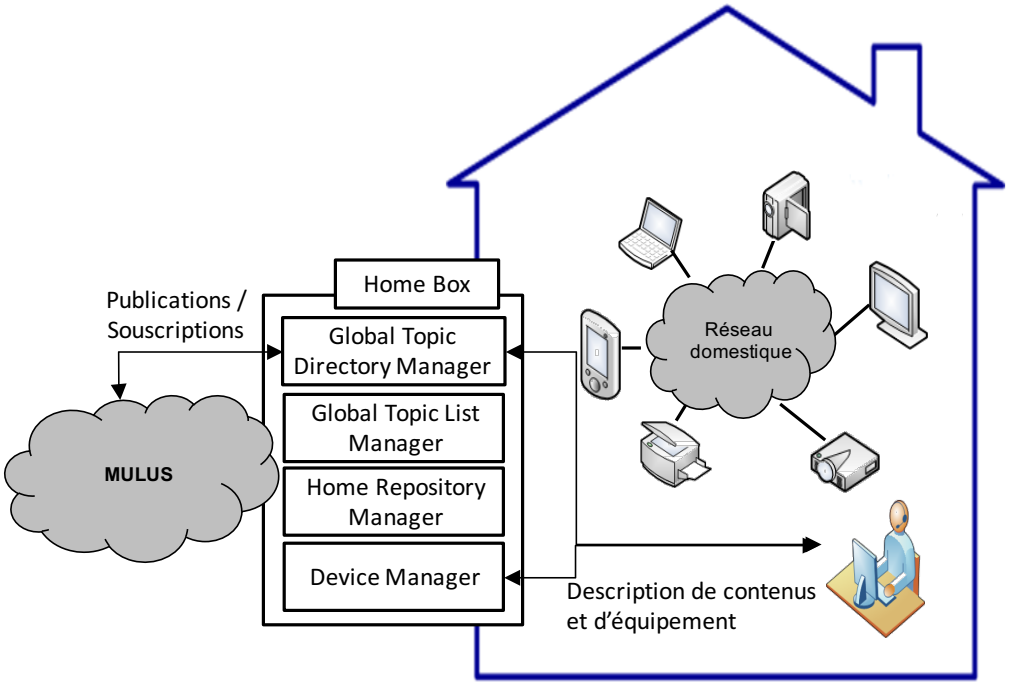


Figure 3.2 – Architecture du *Home Box Entity*

3.4.1.1 *Home Repository Manager* (HRM)

Le *Home Repository Manager* maintient la liste des contenus à partager et qui sont disponibles chez l'utilisateur avec leurs descriptions sémantiques annotées. Les annotations sémantiques sont fondées sur l'ontologie du domaine utilisée (définie dans la section 3.3). Le HRM fournit les classes nécessaires, les propriétés des données et des objets pour décrire tout type de données. Quand un utilisateur publie ou cherche un contenu (un objet multimédia par exemple), il introduit les informations nécessaires qui seront transformées en une règle SWRL. Cette règle est appliquée à l'ontologie du domaine pour déterminer le ou les sujet(s) souhaité(s) par l'utilisateur en utilisant le moteur d'inférence (Pellet). Par la suite, l'utilisateur décide quel sujet sera partagé ou demandé au reste de la communauté. Le HRM envoie subséquemment le sujet final au Global Topic List Manager.

3.4.1.2 *Global Topic List Manager* (GTLM)

Le *Global Topic List Manager* maintient l'ontologie du domaine. Il intervient quand l'utilisateur cherche à envoyer un nouveau sujet (souscription ou publication) dans la communauté globale. Le GTLM est contacté pour vérifier si le sujet choisi par l'utilisateur existe déjà (appartient à l'ontologie du domaine). Si le sujet n'existe pas, il sera ajouté à la liste de GTLM (mise à jour de l'ontologie du domaine). Le GTLM notifie par la suite tous les GTLM des HBE pour déployer et accueillir le nouveau GTLM afin de maintenir ce nouveau sujet.
Après établissement du sujet, le GTLM donne le sujet choisi finalement au GTDM.

3.4.1.3 *Global Topic Directory Manager* (GTDM)

Chaque *Global Topic Directory Manager* est responsable de maintenir un sujet et la liste des souscrits à ce sujet. Chaque GTDM maintient aussi la liste de tous les GTDM qui sont sémantiquement plus généraux du point de vue sujet. Cette liste est utilisée pour router les publications sémantiquement. Le GTDM mémorise aussi la liste des HRM contenant des objets liés au sujet qu'il maintient pour servir au moment du routage. Le principe de routage sera expliqué plus tard dans la section 3.4.3.

3.4.1.4 *Device Manager* (DM)

Le *Device Manager* est utilisé lors de la réception des contenus souhaités au niveau du cluster de l'utilisateur. Il représente un deuxième système publier/souscrire basé contenu et centralisé. Il est responsable de sélectionner le dispositif ayant les caractéristiques exigées pour supporter les données constituant le bundle reçu dans le cluster du réseau domestique. Cette sélection est fondée sur une description des équipements, faite par l'utilisateur, sous forme de souscription lors de l'expression des besoins et sous forme de publication lors de la production de contenus.
Au moment de la souscription, l'utilisateur est guidé pour exprimer ses préférences tout en tenant compte des capacités des dispositifs du point de vue :

- matériel : sur lequel l'application s'exécute : mémoire, CPU, taille d'écran, etc. ;
- plateforme logicielle (système d'exploitation) : sur laquelle les composants multimédias sont hébergés ;

– application : l'application individuelle, comme un navigateur ou un logiciel.

Pour la publication de contenus, l'utilisateur décrit pour chaque objet publié les exigences techniques pour qu'un dispositif soit capable de supporter ces contenus. Le DM détermine par la suite les équipements à utiliser pour chaque contenu reçu en faisant le matching entre les souscriptions et les publications. Ceci nous permet de tenir compte des préférences des utilisateurs et des capacités des dispositifs.

3.4.2 Codage sémantique

Dans cette section, nous définissons la structure de l'ontologie du domaine proposée. Ensuite, nous expliquons notre méthode d'indexation de cette ontologie pour le routage sémantique sur DHT. Enfin, nous expliquons le processus de mise à jour de cette ontologie proposé pour l'ajout de nouveaux concepts.

3.4.2.1 Définition de l'ontologie de domaine

Pour répondre au besoin d'expressivité, nous définissons une ontologie de domaine qui couvre tout le domaine de l'application à gérer (application multimédia par exemple) pour inclure la définition des concepts primordiaux d'une large taxonomie (voir Figure 3.3). Nous concevons cette ontologie de domaine de façon que les classes soient structurées selon une relation d'héritage équivalente à celle utilisée dans la Programmation Orientée Objet. Ainsi, nous obtenons une taxonomie sous forme d'arbre dont la racine présente le concept le plus général qui décrit le domaine de l'application. En descendant vers les fils, nous trouvons les concepts les plus spécifiques qui héritent du concept père. Le GTLM du HBE est responsable de maintenir cette ontologie de domaine.

Dans le contexte du multimédia par exemple, la racine de l'arbre représente la classe Multimédia comme étant la classe la plus générique dans le domaine. En descendant vers les feuilles, nous trouvons les classes des sujets qui sont plus spécifiques. Ainsi, quand un utilisateur est intéressé par le sujet Video (à titre d'exemple) et un producteur publie un évènement lié au sujet Film, cette structure d'arbre permet de détecter que le Film est une Video et par la suite le/les souscrits à ce sujet seront notifiés. De cette manière, les nœuds DHT qui sont responsables des sujets des classes primaires (*child topic*) doivent connaître et communiquer avec les autres nœuds responsables des sujets plus généraux (*parent topic*).

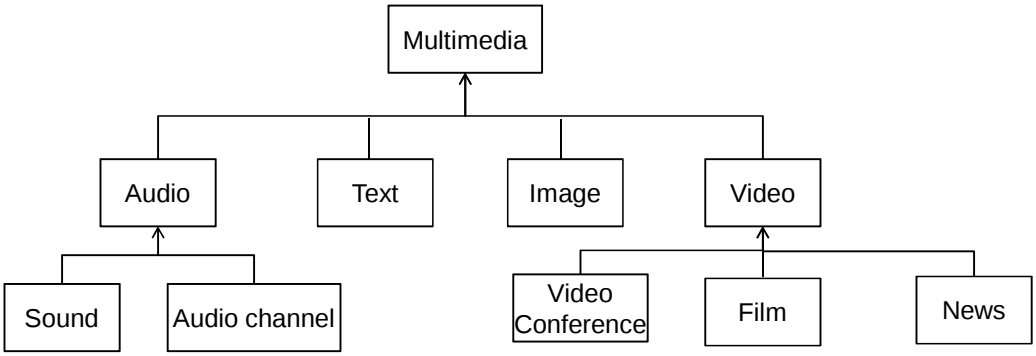


Figure 3.3 – Branche de l'ontologie du domaine multimédia

3.4.2.2 Indexation de l'ontologie de domaine avec les nombres premiers

Nous définissons une méthode d'indexation de l'ontologie de domaine fondée sur les nombres premiers. Cette méthode permet d'attribuer un nombre premier ou une multiplication de nombres premiers comme identifiant sémantique pour chaque classe de l'ontologie. Cet identifiant permet de reconnaître les classes ayant une relation sémantique d'équivalence ou d'héritage par le calcul des diviseurs de l'identifiant qui sont des nombres premiers. Il est décrit dans la partie propriété qui caractérise la classe correspondante en OWL. Il est utilisé par chaque GTDM pour reconnaître les GTDM qui sont sémantiquement plus généraux du point de vue sujet.

Comme montré par la Figure 3.4, nous commençons par attribuer le premier nombre premier (2) à la racine. En descendant dans le niveau suivant, nous multiplions l'identifiant de la classe mère par le nombre premier suivant pour avoir l'identifiant de chaque classe.

Ainsi, la recherche des nombres premiers diviseurs de l'identifiant d'une classe permet de déterminer les relations d'héritage avec les autres classes [70]. Par exemple, pour la classe d'identifiant 30, nous cherchons les nombres premiers diviseurs qui sont 2, 3 et 5. La classe mère est alors d'identifiant 2*3=6 déterminé par la multiplication de tous ses diviseurs sauf le plus grand.

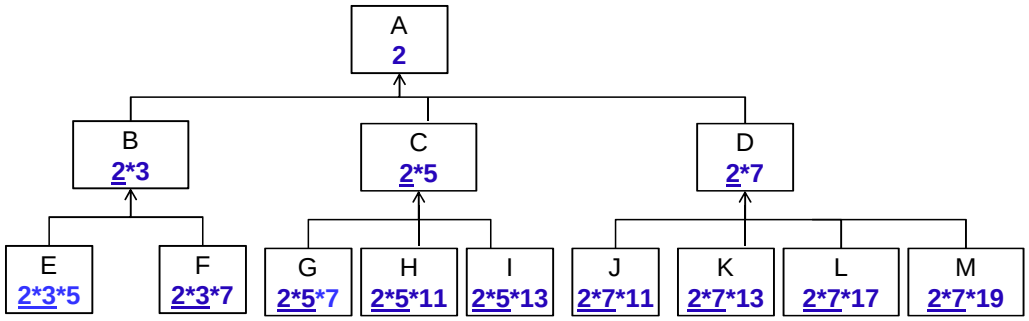


Figure 3.4 – Codage sémantique de l'ontologie de domaine

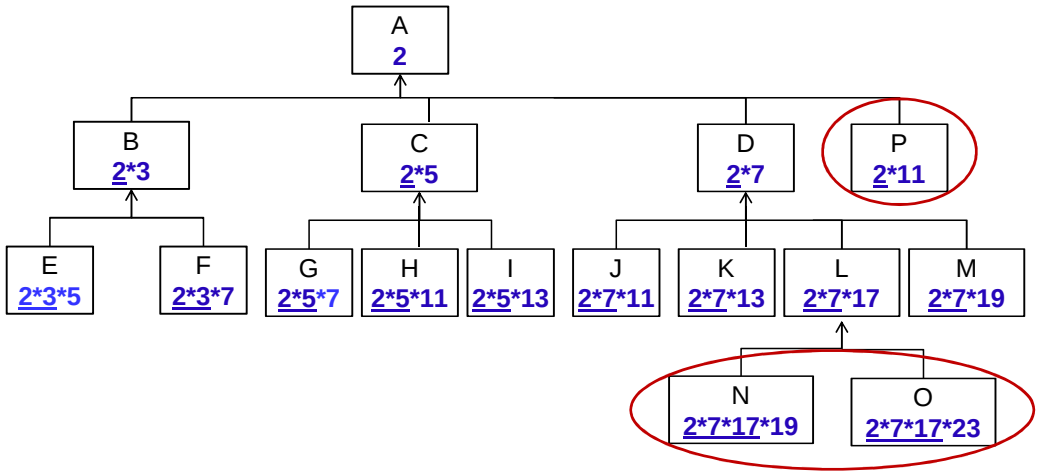
3.4.2.3 Mise à jour des index de l'ontologie de domaine

La définition d'une ontologie de domaine permet de couvrir tous les concepts d'un domaine donné. L'apparition de nouveaux concepts est rare mais possible. Si ces concepts paraissent pertinents et souvent utilisés alors ils doivent être ajoutés à notre ontologie de domaine.

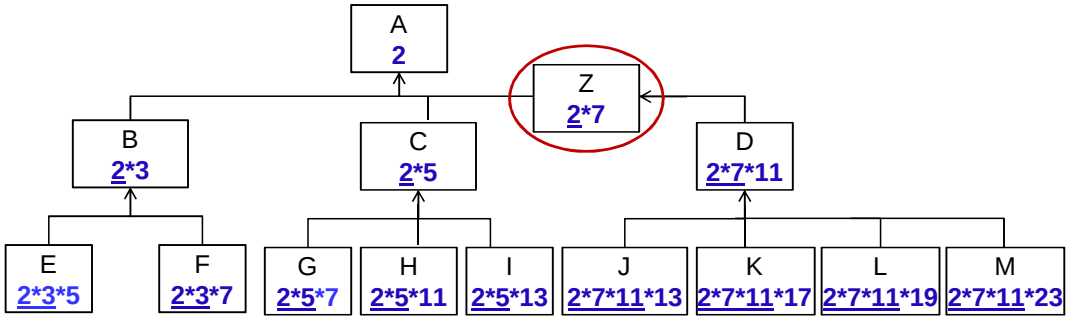
Le processus de mise à jour de l'ontologie de domaine est un peu délicat. D'une part, l'emplacement du nouveau concept dans l'ontologie de domaine influence la fonction de base de notre RS (répondre aux besoins exacts des utilisateurs). D'autre part, nous devons mettre à jour toutes les copies chez les GTLM des différents pairs pour leur synchronisation.
Pour l'ajout d'un nouveau concept, nous avons deux cas possibles :

- ajout de concept fils : le nouveau concept s'ajoute comme un fils (feuille) dans l'arbre de l'ontologie de domaine. L'indexation se fait en respectant les identifiants du nœud père et des nœuds voisins attachés au même père. Il prend l'identifiant du père et le multiplie par le nombre premier qui le suit et qui n'est pas encore utilisé par les voisins. Ceci est montré par la Figure 3.5(a). Pour les concepts N et O, ils se rattachent au père L (d'identifiant 2*7*17) qui n'a pas de fils, alors ils prennent les identifiants 2*7*17*19 et 2*7*17*23, respectivement. Pour le concept P, il se rattache au père A et il prend l'identifiant de A multiplié par le nombre premier suivant non utilisé par les voisins B, C et D.
- ajout de concept parent : le nouveau concept peut être plus générique que les autres concepts alors il s'ajoute entre un père et un fils de l'arbre de l'ontologie de domaine en respectant toujours la relation d'héritage. Il prend alors l'identifiant de son père

multiplié par le nombre premier suivant et engendre la modification des identifiants de tous ses fils. Ceci est montré par la Figure 3.5(b). Pour le concept Z, il se rattache entre le père A et le fils D d'identifiant 2*7 auparavant. Il prend alors l'identifiant de son fils (2*7). En descendant dans le niveau, le reste de ses concepts fils (D, J, K, L et M) prend l'identifiant de Z multiplié par les nombres premiers suivants avec le même principe que notre méthode d'indexation expliquée dans la section 4.9.



(a) Ajout de concepts fils



(b) Ajout de concept parent

FIGURE 3.5 – Mise à jour de l'ontologie de domaine

Dans la section suivante, nous utilisons l'indexation de l'ontologie de domaine pour détailler la communication entre les différents pairs (HBE) de notre plateforme MULUS.

3.4.3 Communication entre les HBE

La communication entre les HBE est établie souvent pour l'acheminement des souscriptions et des publications ou pour la mise à jour de l'ontologie de domaine et leur inter-synchronisation.

3.4.3.1 Phase de souscription

Après la définition et l'indexation de l'ontologie du domaine au niveau des GTDM, nous détaillons la communication entre les HBE des différents clusters en utilisant le codage de l'ontologie du domaine avec les nombres premiers.

Pendant la phase de souscription, les identifiants de la classe correspondante au sujet de l'intérêt (déterminé par l'ontologie du domaine et la règle SWRL) servent comme clé de la souscription à router vers le nœud responsable. Cette phase reste indemne pour un système publier/souscrire basé sujet et de topologie P2P structurée.

3.4.3.2 Phase de publication

Pour la phase de publication, la notification doit atteindre les souscrits au sujet reconnu par l'identifiant de la classe ainsi que tous les souscrits aux sujets plus génériques. Ainsi, quand la publication arrive au GTDM responsable, ce dernier traite son ID en calculant tous les diviseurs, qui sont des nombres premiers, comme expliqué par la Figure 3.6.

- Si l'identifiant du sujet de publication correspond à un nombre premier, alors cette publication concerne un sujet générique. Une notification est alors envoyée aux souscrits à ce sujet uniquement et la phase de publication s'achève. Dans ce cas, nous retrouvons que l'identifiant du sujet est égal à 2 et que l'identifiant du nœud est 0x0002.
- Si l'identifiant est une multiplication de nombres premiers, alors les diviseurs calculés permettent de retrouver les identifiants des classes mères. Par la suite, le GTDM responsable notifie les souscrits retrouvés localement et dissémine cette publication vers tous les responsables des classes mères.

L'arbre à droite de la Figure 3.6 montre la communication entre les HBE responsables des classes de l'ontologie de domaine en utilisant les nombres premiers. Les arbres à gauche montrent les arbres de multicast de chaque HBE responsable d'un sujet (classe de l'ontologie de domaine). Dans cet exemple, le sujet publié à partir du nœud 0x11B2 arrive au nœud

responsable 0x258E. Ce dernier utilise donc son arbre de multicast (formé par les chemins des souscriptions) pour la notification des utilisateurs (0x02A3 et 0x1111). Puisque l'identifiant de 0x258E est une multiplication de nombres premiers (2, 11, 19 et 23), la publication est envoyée au nœud responsable du sujet de la classe mère dont l'identifiant est 0x01A2 (=2*11*19). Ce dernier procède de la même façon pour notifier ses souscrits (par son arbre de multicast) et envoie cette publication au nœud responsable de la classe mère.

Pour traiter le cas où l'utilisateur souscrit souhaite que les contenus attendus soient partagés uniquement par leurs amis, nous vérifions alors que la source du contenu est parmi les amis de cet utilisateur.

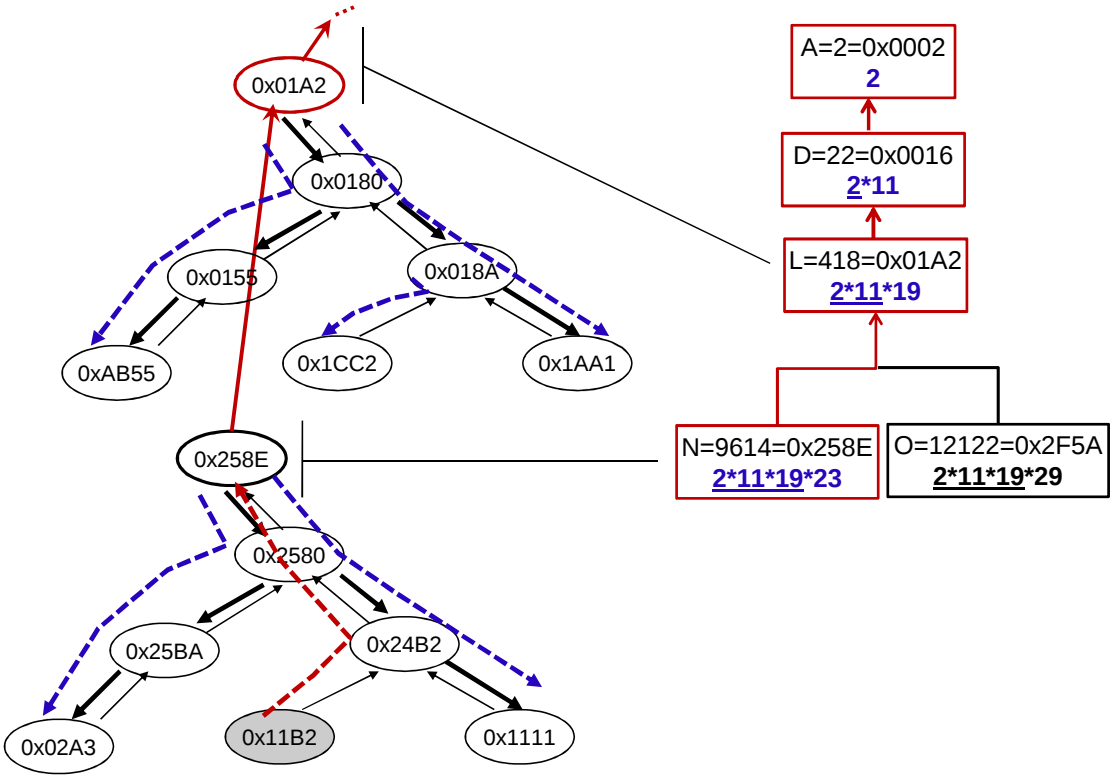


Figure 3.6 – Phase de publication avec routage fondé sur les nombres premiers

3.4.3.3 Mise à jour de l'ontologie de domaine

Pour la mise à jour de l'ontologie de domaine, une notification est envoyée par le GTLM responsable de cette mise à jour à tous les autres GTLM pour se synchroniser. Cette mise à jour peut influencer aussi l'emplacement des souscriptions maintenues par les GTDM. Comme nous l'avons expliqué, la mise à jour peut consister à l'ajout d'un concept fils ou

d'un concept parent.
L'ajout d'un concept fils n'influence pas l'emplacement des souscriptions déjà mémorisées car les identifiants des concepts initiaux ne changent pas (voir Figure 3.5(a)).
Pour l'ajout de concept parent, la mise à jour de l'ontologie de domaine influence l'emplacement des souscriptions car les identifiants des classes initiales changent. Par la suite, les GTDM responsables des sujets des classes filles de la nouvelle classe transmettent leurs listes des souscrits vers les nouveaux GTDM selon les nouveaux identifiants (voir Figure 3.5(b)).

3.5 Exemple de scénario : partage de données multimédias

Notre solution peut être utilisée dans plusieurs contextes, en particulier pour le partage de données multimédias. En effet, l'usage fréquent d'équipements ayant une capacité de communication, a contribué à l'apparition des scénarios de multimédias ubiquitaires où les utilisateurs peuvent facilement créer des sessions de multimédia *ad hoc*. De plus, l'utilisation croissante des RS a contribué à l'apparition de nouveaux scénarios avec de nouveaux modes de partage et de communication des éléments multimédias. Avec cette large diversité de scénarios multimédias ubiquitaires, nous avons trouvé que les scénarios multimédias dans un réseau domestique sont particulièrement importants vu que les utilisateurs produisent, consomment et partagent une quantité de multimédias importante à partir de leurs réseaux domestiques. Les forums d'UPnP [9] et de DLNA [10] ont montré quelques scénarios et ont proposé un ensemble de technologies pour faciliter l'intégration des équipements domiciles dans l'expérience de la vie numérique [73, 74, 75, 76]. Les utilisateurs peuvent aussi produire, consommer et partager des éléments multimédias quand ils sont à l'extérieur de leurs réseaux domiciles. En outre, les utilisateurs s'attendent aussi à une bonne qualité de service de leur RS quand ils sont dans leurs réseaux domestiques. Les concepts étendus traitent des scénarios où les services à domicile sont accessibles à l'utilisateur s'il est à l'intérieur ou à l'extérieur de son propre foyer (bureau de travail, maison, etc.). Dans un contexte plus large, l'utilisateur n'a pas accès à ses services seulement à partir de sa voiture, son bureau ou autres endroits, mais il cherche à profiter des services, peu importe son emplacement. Des projets comme

9. Universal Plug and Play http ://www.upnp.org/
10. Digital Living Network Alliance http ://www.dlna.org/

Feel@Home[11] et Encompas [77] ont ciblé et développé ce genre de scénarios.

Nous avons identifié deux scénarios dans lesquels une description sémantique (basée ontologie) aide à la recherche des contenus multimédias selon les préférences de l'utilisateur. Dans le premier scénario, présenté dans la Figure 3.7, l'utilisateur fournit une description du sujet auquel il s'intéresse selon l'ontologie du domaine (1). Une fois que l'utilisateur a choisi ses préférences, cette information est utilisée par le raisonneur avec un ensemble de règles afin de fournir une liste de sujets qui satisfont la demande de l'utilisateur (2). Les résultats obtenus par le raisonneur et les règles sont présentés à l'utilisateur pour qu'il sélectionne l'élément multimédia désiré (3).

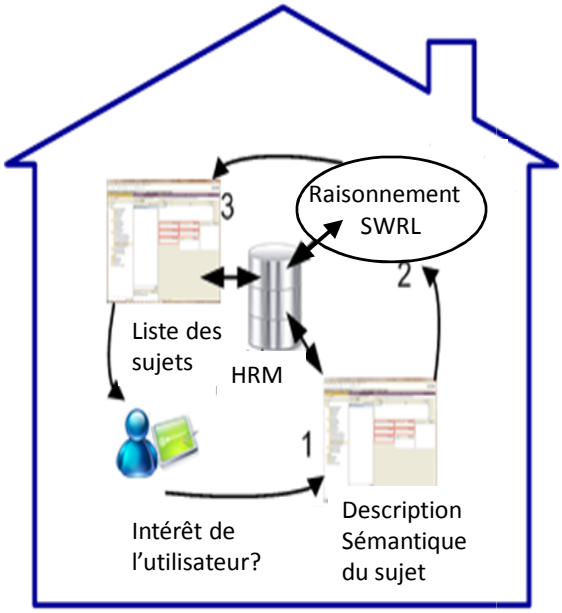


Figure 3.7 – Scénario réseau domestique

Dans le deuxième scénario, illustré par la Figure 3.8, l'utilisateur souhaite faire une recherche plus large pour inclure les contenus multimédias trouvés non seulement dans son cluster (foyer ou domicile) mais aussi dans d'autres clusters distants. Dans ce cas, la solution que nous avons proposée aide à trouver tous les sujets partagés par les autres utilisateurs des clusters distants.

11. www.nics.uma.es/projects/feelathome

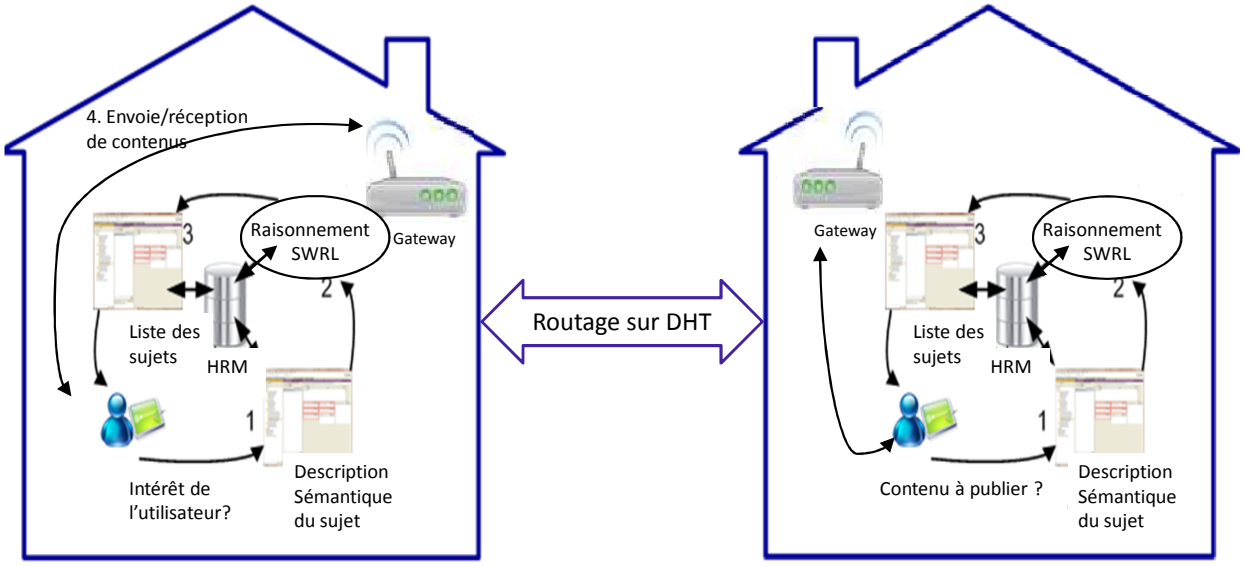


Figure 3.8 – Scénario partage entre deux réseaux domestiques

3.6 Évaluation de performance : réduction du trafic

Pour évaluer notre approche, nous l'avons implémentée sur le système Scribe [20]. L'étude expérimentale a été faite sur le simulateur FreePastry en utilisant une machine Linux dotée d'un processeur i5 CPU 2.53 GHz et 4 GB de RAM.

Nous rappelons que notre but est de répondre aux besoins exacts des utilisateurs sur les RSD en P2P et de réduire le trafic sur le réseau Internet. Ce dernier objectif est aussi un résultat direct du premier. En effet, éviter d'envoyer des données inutiles aux utilisateurs implique nécessairement la réduction considérable du trafic. Pour le prouver, nous avons testé avec deux scénarios. Dans le premier scénario, nous injectons 4000 publications sans utiliser le système publier/souscrire. Dans le deuxième scénario, nous injectons 1000 souscriptions sémantiques et les mêmes 4000 publications. Ces deux scénarios permettent de comparer un RS fondé sur un système publier/souscrire par rapport aux réseaux existants.
Le résultat montre que le nombre de notifications envoyées aux utilisateurs est remarquablement réduit en utilisant MULUS (avec publier/souscrire). En effet, l'acheminement des publications peut atteindre 100% si nous considérons le cas de Facebook où le partage se fait entre tout un groupe d'amis. Ce type de partage ne respecte pas les intérêts des utilisateurs et par conséquent ces derniers seraient insatisfaits. Cependant, en considérant les intérêts

des utilisateurs (avec MULUS), la réduction peut atteindre environ 82% comme montré par la courbe de la Figure 3.9.

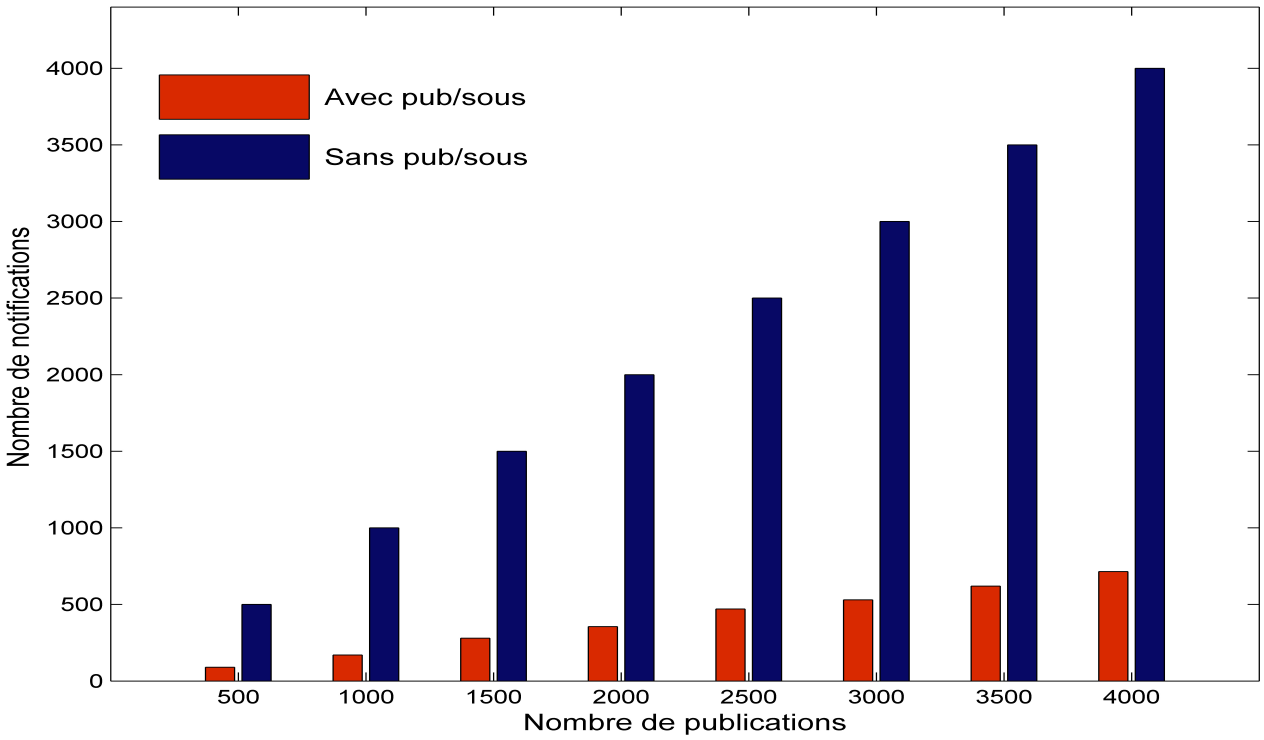


Figure 3.9 – Réduction du trafic par un système publier/souscrire P2P sémantique

3.7 Conclusion

Dans ce chapitre, nous avons étudié la sémantique dans les réseaux publier/souscrire fondé sur la DHT. Nous avons distingué trois types d'approches : le premier type assure la scalabilité grâce à la tologie P2P structurée mais qui n'améliore pas l'expressivité. Le deuxième type d'approches essaye d'équilibrer entre la scalabilité et l'expressivité. Il améliore peu l'expressivité mais il n'assure pas la scalabilité vu la topologie hiérarchique de son service d'évènements. Le troisième type d'approches améliore l'expressivité et même utilise l'ontologie pour assurer plus de sémantique. Cependant, il n'est pas scalable à cause de la topologie centralisée. Nous avons constaté que l'utilisation de la sémantique avec un système publier/souscrire basé DHT est un défi. D'une part, la sémantique permet d'utiliser des expressions sémantiquement équivalentes mais syntaxiquement différentes. D'autre part,

le routage avec DHT est fondé sur les clés calculées par le hachage des données à partager. Ainsi, la différence syntaxique possible avec la sémantique perturbe le routage des messages sur la DHT.
Pour résoudre ce problème, nous avons proposé d'intégrer une entité logicielle appelée HBE pour MULUS. Cette entité est déployée dans chaque réseau domicile pour assurer les fonctionnalités sollicitées pour la gestion interne des clusters et le partage des sujets avec les autres serveurs d'évènements. Cette entité maintient une ontologie du domaine structurée (hiérarchique) et partagée entre tous les utilisateurs. Nous avons proposé ensuite une méthode d'indexation de cette ontologie fondée sur les nombres premiers. Cette méthode offre un routage sémantique (pour communiquer entre les utilisateurs distants) à travers la DHT et permet par conséquent de répondre aux intérêts des utilisateurs.
Enfin, pour valider notre approche, nous avons conduit des tests expérimentaux pour montrer que l'intégration d'un système publier/souscrire sémantique réduit considérablement le trafic sur le réseau.

Pour mieux répondre aux besoins des utilisateurs de notre plateforme MULUS et pour améliorer le degré d'expressivité de MULUS, nous utiliserons la composition d'évènements dans le chapitre suivant. En effet, bien que la sémantique avec les ontologies améliore beaucoup l'expression des intérêts, elle ne traite pas les intérêts composites.

Chapitre

4 Composition d'évènements sur MULUS

4.1 Introduction

Ce chapitre adresse la composition de contenus partagés sur notre plateforme MULUS. Avec la composition d'évènements, le filtrage strict permet d'éviter la diffusion de messages inutiles ne répondant pas aux contraintes temporelles et logiques imposées par les utilisateurs. Le traitement des intérêts composites est assez important dans les RS puisqu'il permet d'alléger l'utilisation de la bande passante sur le réseau Internet. Nous cherchons à traiter la composition d'intérêts au niveau du système publier/souscrire, que nous utilisons comme couche principale de notre plateforme MULUS.
La gestion d'évènements composites et leur routage sont largement explorés dans la littérature mais ils nécessitent encore des améliorations. En effet, les solutions proposées souffrent du problème de scalabilité et d'expressivité au niveau des relations de compositions traitées.

Notre objectif est de fournir une approche scalable pour la gestion et le routage des évènements composites avec toutes les relations de compositions possibles sur un système publier/souscrire P2P structuré. A cette fin, nous proposons une structure d'indexation à trois dimensions que nous appelons CECube. Cette structure permet de gérer la composition d'évènements sur un réseau de brokers de topologie P2P structurée. CECube permet de vérifier toutes les relations logiques et temporelles possibles et élimine par la suite tous les transferts inutiles.

Dans la suite de ce chapitre, nous commençons par présenter les concepts de base pour expliquer ensuite le problème de gestion des évènements composites. Ensuite, nous détaillons les travaux existants pour identifier les points faibles à considérer. Dans la section suivante, nous détaillons notre solution pour la gestion distribuée des évènements composites en P2P. Enfin, nous validons notre proposition par des résultats expérimentaux.

4.2 Concepts de base

Dans cette section, nous détaillons quelques concepts de base pour mieux comprendre le problème de gestion des évènements composites.

4.2.1 Définition d'un évènement

Nous trouvons dans la littérature plusieurs définitions de la notion d'évènement selon le domaine d'application. Parmi les définitions les plus courantes, un évènement est défini comme étant l'occurrence d'une situation d'intérêt dans un système, ce qui nécessite généralement une réponse automatique par le système [78].

Cette notion est explorée dans les systèmes publier/souscrire dis encore systèmes basés évènements. Dans ces systèmes, l'évènement est équivalent à une production d'information sur le réseau qui stimule la réaction des serveurs d'évènements pour faire le filtrage et le routage de cette information [79]. L'occurrence d'un évènement se traduit par une notification (*event notification*) sous forme d'un message envoyé aux souscrits pour répondre à des intérêts exprimés auparavant. Ces intérêts peuvent être simples (primitifs ou atomiques) ou composites. D'où l'apparition de deux notions : évènement primitif et évènement composite.

4.2.2 Évènement primitif

Les évènements primitifs (ou atomiques) sont les évènements considérés atomiques et instantanés. Ils expriment un intérêt simple et non lié à d'autres évènements.

4.2.3 Évènement composite

Un évènement simple peut être relatif à d'autres évènements par une ou plusieurs relations temporelles ou/et logiques. Ceci résulte en l'apparition de l'évènement composite. Les évènements composites sont les évènements de notification qui surviennent suite à l'arrivée d'autres évènements, dans un ordre ou un motif bien déterminé.
Par exemple, nous pouvons détecter, chaque fois, qu'un évènement de type T_2 se produit directement, après la survenance d'un évènement de type T_1 (une séquence de T_1 et T_2), ou lorsqu'un évènement de type T_1 ou un évènement de type T_2 se produit (une disjonction de

T_1 et T_2). La détection de ces modèles d'évènements composites peut être traduite par la production d'une notification d'évènement composite représentant l'occurrence particulière d'une suite d'évènements.

4.2.4 Souscription composite

Nous considérons la définition de souscription composite comme proposé dans [80] : "Une souscription composite consiste à des souscriptions reliées par des opérateurs logiques et temporels. Elles peuvent être utilisées pour exprimer un intérêt à un évènement composite. Une souscription composite n'est matchée que si toutes les souscriptions atomiques qui la composent sont vérifiées".

Dans de ce manuscrit, nous appelons :

- évènement primitif : tout évènement atomique n'ayant pas de relation avec d'autres évènements ;
- évènement composite : une souscription composite ou un évènement de notification composite ;
- évènement sous-composite : un élément d'un évènement composite qui peut être primitif ou composite.

4.3 Problématique

Comme montré par la Figure 4.1, plusieurs évènements sont collectés dans le service d'évènements pour être regroupés selon les souscriptions des utilisateurs. Ainsi, la cohérence de regroupement des évènements demande un effort de coordination entre les serveurs d'évènements formant ce service. Si les évènements sont mal regroupés et mal filtrés, les utilisateurs seront submergés par des publications inutiles.

Pour mieux comprendre les problèmes soulevés lors du traitement d'évènement composite, nous considérons l'exemple montré par la Figure 4.1 avec un service d'évènements centralisé. À gauche, nous représentons les souscriptions (composites/atomiques) exprimant les intérêts des utilisateurs. Les publications sont représentées à droite selon leur ordre de réception. Au milieu, nous représentons le service d'évènements responsable de la mémorisation des souscriptions, de leur matching avec les publications et du routage des publications selon le résultat de matching obtenu.

Dans cet exemple, S_0 est une souscription qui relie e_2 et e_3 avec la relation logique *XOR*. Ainsi, si le service d'évènements reçoit E_3 suivi par E_2 qui relient e_3 et e_2 respectivement, alors il doit notifier par E_3 et ignorer E_2. Vu que E_2 relie S_2, elle est alors envoyée au(x) utilisateur(s) souscrit(s) à S_2. Pour traiter S_1, le service d'évènements doit vérifier les relations logiques et temporelles pour former l'évènement composite correspondant. En recevant E_6, il doit vérifier la réception de E_5 auparavant pour pouvoir vérifier la relation temporelle (temps d'arrivée de $E_6 \geq$ temps d'arrivée de E_5). Sinon, les utilisateurs seront bombardés par un grand nombre d'évènements primitifs dont plusieurs sont inutiles et doivent alors être filtrés avant d'être acheminés aux utilisateurs. Ainsi, la composition d'évènements permet de mieux satisfaire les intérêts des utilisateurs et de réduire le trafic sur le réseau en se débarrassant des évènements inutiles sur le réseau.

Le problème de gestion devient plus complexe si les évènements sont collectés par un service d'évènements distribué (c'est-à-dire formé par des serveurs d'évènements dispersés géographiquement) et si ces évènements doivent être filtrés, combinés et routés selon différents modèles de compositions.

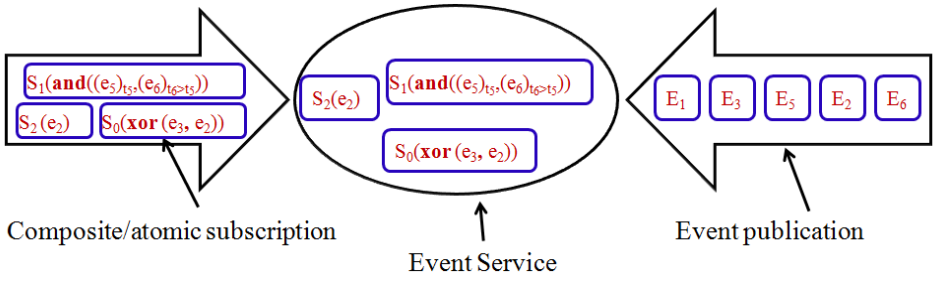


FIGURE 4.1 – Gestion des évènements composites

Le problème de la gestion et du routage des évènements composites a été exploré dans les systèmes publier/souscrire et demande des efforts supplémentaires pour être amélioré dans les systèmes d'architecture P2P structurée. Nous commençons par une étude des travaux existants dans une première section (4.4). Puis, nous détaillons l'approche proposée dans une deuxième section (4.5).

4.4 État de l'art : composition d'évènements dans les systèmes publier/souscrire

A notre connaissance et comme annoncé dans le premier chapitre, le problème de composition de données ou d'intérêts sur les RS existants n'est pas encore traité. Par conséquent, nous nous intéressons dans ce qui suit aux travaux qui adressent le problème de la composition d'évènements dans les systèmes publier/souscrire, puisque nous avons conçu notre plateforme MULUS autour de ce type de systèmes.
Les solutions existantes peuvent être classées en trois catégories : les solutions centralisées, les solutions distribuées et les solutions P2P (DHT).

4.4.1 Composition d'évènements dans les systèmes publier/souscrire centralisés

Les solutions centralisées sont apparues en premier et sont plus efficaces que les solutions distribuées en termes de rapidité de matching et d'expressivité (les relations de compositions vérifiées). En effet, ces solutions utilisent un service d'évènements centralisé qui est considéré comme étant le cœur du système et il est responsable du filtrage, composition et routage des évènements reçus de plusieurs producteurs et envoyés vers plusieurs consommateurs. Dans la suite, nous détaillons les solutions centralisées afin de déterminer leurs avantages et leurs inconvénients.

4.4.1.1 Système RUBCES

RUBCES [12] est un système fondé sur les règles de compositions d'évènements. Il définit un service d'évènements centralisé pour la gestion et le stockage des souscriptions. La Figure 4.2 représente l'architecture du service d'évènements. Ce système contient essentiellement une base de règles (Rule Base) pour la mémorisation des règles de compositions et un module de gestion (Event Manager) pour le matching des évènements. Les souscriptions reçues sont mémorisées par l'entité Rule Base désignée comme une base de données centralisée contenant des tables pour arranger les souscriptions en entrée. Quand une publication est reçue, l'Event Manager contrôle la liste de souscriptions pour détecter les souscriptions qui correspondent à cette publication. Dans le cas de matching avec un élément d'une souscrip-

tion composite, cette publication est transmise à l'entité Notification Queue pour attendre le reste des évènements.

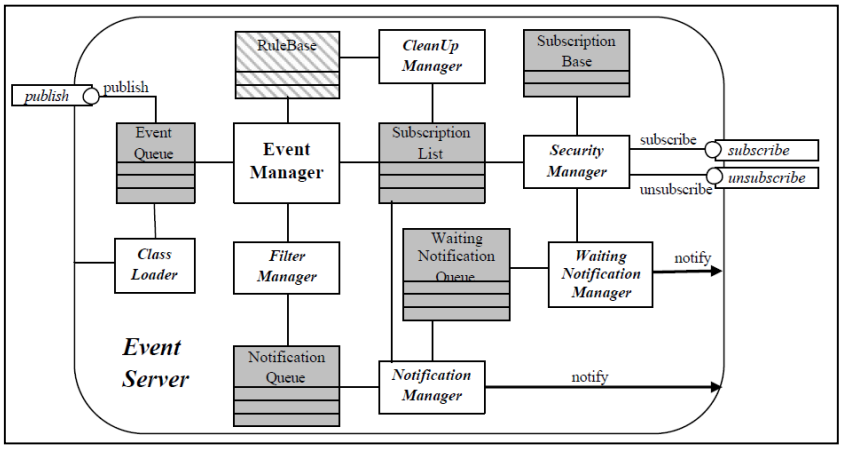


FIGURE 4.2 – Architecture détaillée du serveur d'évènements de RUBCES [12]

RUBCES permet le matching des évènements primitifs et composites avec seulement des relations logiques (AND/OR) définies par des règles. De plus, son architecture centralisée peut causer une surcharge du service d'évènements et réduire son taux de disponibilité. Avec la possibilité de panne du serveur, la qualité de service se dégrade, ce qui engendre une éventuelle insatisfaction des utilisateurs. En outre, RUBCES ne supporte pas les contraintes et les relations temporelles entre les évènements.

4.4.1.2 Détection des évènements composites par les automates à états finis (FSA)

P.R. Pietzuch *et al.* [13] proposent un framework pour la détection des évènements composites. L'architecture proposée du service d'évènements composite est formée par 3 couches comme illustré par la Figure 4.3. La couche *Application* soumet les souscriptions composites à la couche de détection (*CE Detection Framework*) responsable de déterminer les patterns de composition grâce au module de détection *CE detector*. Ce dernier est fondé sur les FSA conçus pour détecter les modèles des évènements composites. La couche de détection communique avec la couche publier/souscrire pour traiter le matching des évènements atomiques. La couche *Application* peut aussi communiquer directement avec la couche publier/souscrire dans le cas où les évènements sont simples.

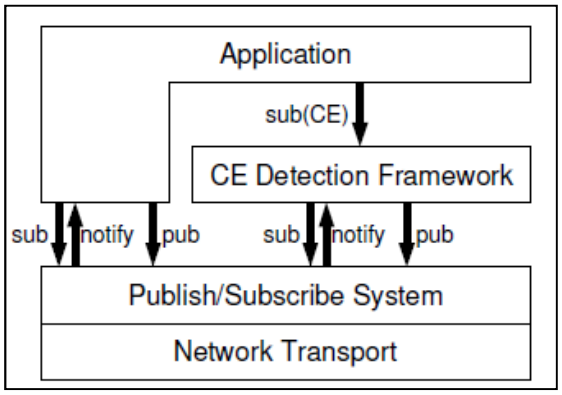


Figure 4.3 – Framework de détection d'évènements composites [13]

Il est vrai qu'avec les FSA, le framework proposé permet de définir des modèles de compositions variés avec des contraintes temporelles riches et de détecter les répétitions de modèles. Mais, les FSA ne peuvent pas être déployés sur un réseau distribué. En effet, un problème de synchronisation d'horloge entre les nœuds se pose d'autant plus que les FSA sont sensibles aux contraintes temporelles. De plus, ce framework ne traite pas les relations logiques pour la composition d'évènements.

4.4.1.3 Composition d'évènements dans un système de communication multimédia

M. Gao *et al.* [14] ont porté intérêt aux systèmes de communication multimédia utilisant des capteurs distribués sur différents endroits. L'architecture du système proposé est montrée par la Figure 4.4. Tous les évènements composites sont pré-enregistrés sous forme de requêtes. Le moteur de requêtes est formé de trois composants principaux : fenêtre de prédicat (Window Predicate), filtre spatial et composite event matching. L'entrée du système est un flux d'évènements atomiques AES (*Atomic Event Stream*) recueillis par les capteurs. Si un évènement composite (présenté par une fenêtre) manque un prédicat de la fenêtre alors un délai d'attente approprié lui est attribué. Si la composition est vérifiée, elle passe au filtre spatial pour la vérification des contraintes et enfin aux processus de matching d'évènements composites qui génèrent la séquence d'évènements finaux à partir des évènements reçus en entrée.

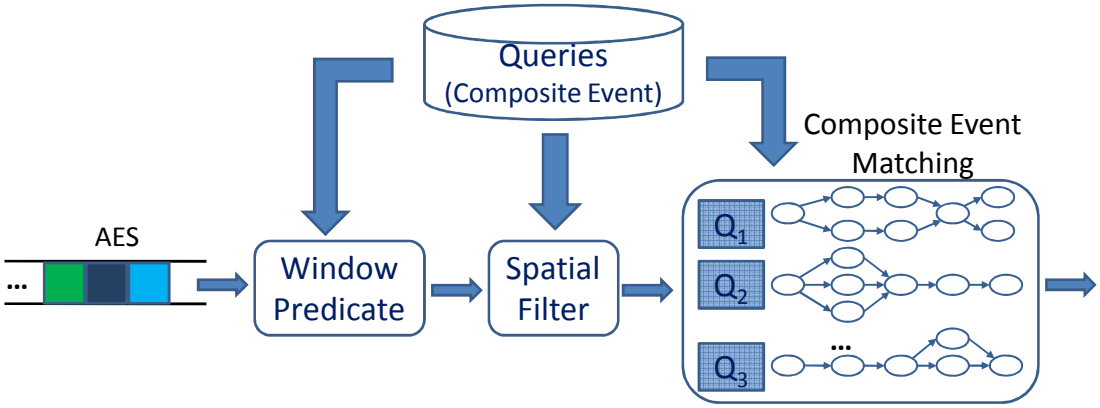


Figure 4.4 – Architecture du système de traitement de composition spatio-temporelle [14]

Cette solution connaît un problème de scalabilité vu que la réception d'un grand flux d'évènements peut engendrer la congestion du système centralisé.

4.4.1.4 PSware pour les réseaux de capteurs sans fil

Steven Lai *et al.* [15] traitent la détection des évènements composites pour les réseaux de capteurs avec des contraintes logiques, temporelles et spatiales. PSware est une application de monitoring avec un réseau de capteurs. Chaque nœud capteur est programmé pour détecter un modèle de composition bien déterminé vu la mémoire limitée des capteurs. Le modèle de composition est défini par un programme image (byte code), résultat de la compilation de l'Event Definition Language (EDL) proposé. EDL est une combinaison de SQL et d'un langage orienté objet. Il permet de supporter 2 types d'opérateurs temporels (séquentiel et parallèle) et 3 opérateurs logiques (conjonction, disjonction et négation). Comme le montre la Figure 4.5, le programmeur de l'application définit des souscriptions en EDL pour être compilées par la suite et pour générer donc du byte code et un module Event Receiving. Ce module est un programme Java exécuté à déployer sur le terminal pour interpréter les données envoyées par les nœuds capteurs et notifier le programmeur de l'évènement survenant. Le byte code généré reflète des programmes images dont chacun présente un modèle de composition. D'autre part, il est figé sur un nœud capteur.

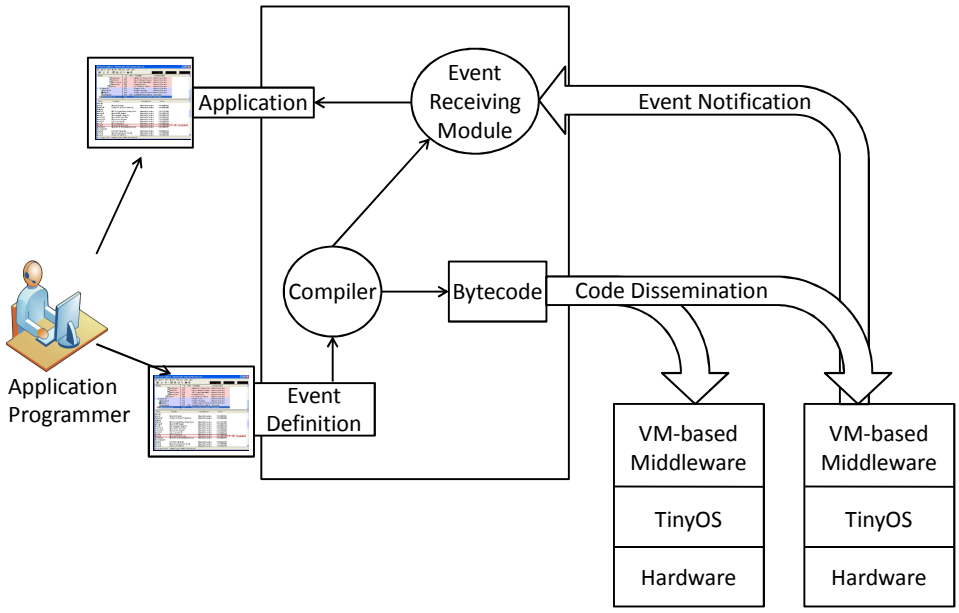


Figure 4.5 – Architecture du système PSWare [15]

Le premier inconvénient est que la détection des évènements composites est réduite aux réseaux de capteurs. Le deuxième inconvénient est que les modèles de compositions sont figés sur les capteurs et ne peuvent pas être modifiés par un simple utilisateur sauf si ce dernier est capable de modifier les programmes sur les capteurs. Ainsi, quand le besoin d'une nouvelle composition s'impose selon les intérêts des utilisateurs, un simple utilisateur ne peut l'intégrer que par l'intervention du programmeur de l'application à travers l'interface dédiée. En outre, cette approche peut être utilisée seulement avec un réseau de capteurs déployés sur une zone limitée comme un aéroport (par exemple, pour contrôler le trafic des vols et des voyageurs), ou une entreprise (pour régler la température selon les horaires de travail).

4.4.2 Détection d'évènements composites avec des FSA distribués

Avec le travail de P.R. Pietzuch *et al.* [13] détaillé dans la section précédente, le CE detector souffre essentiellement du problème de scalabilité avec les automates centralisés. Les mêmes chercheurs suggèrent alors d'améliorer leur solution par la duplication de ces détecteurs dans les endroits favorables sur le réseau selon la bande passante du réseau et selon les sources des évènements composites. Pour améliorer le taux de disponibilité et l'efficacité des détecteurs, les CE detectors doivent être distribués proches des sources des évènements

composites comme expliqué par la Figure 4.6. Même s'il n'y a pas un détecteur de CE pour une nouvelle souscription composite, un nouveau détecteur est défini et soumis sur le serveur d'évènements le plus proche de la source.

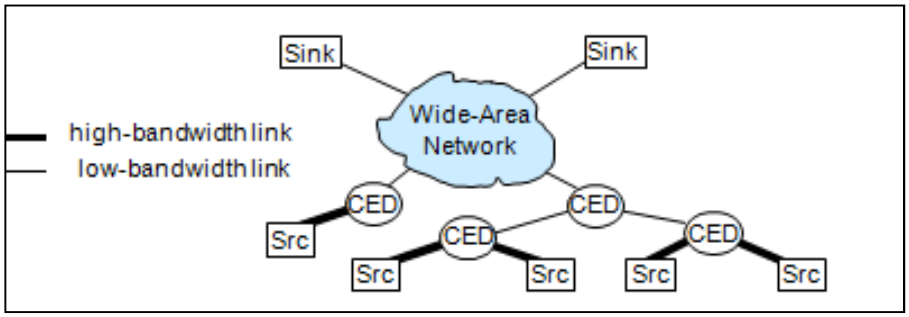


Figure 4.6 – Distribution des détecteurs d'évènements composites [13]

Ceci résout le problème de centralisation mais le problème de scalabilité persiste toujours. Pour la détection de nouvelles compositions, le programmeur doit intervenir pour définir un nouveau FSA et l'intégrer dans les différents détecteurs.

4.4.3 Composition d'évènements sur les systèmes publier/souscrire basés DHT

Le premier travail proposé pour traiter les évènements composites sur un système publier/souscrire basé DHT est celui de Courtenage *et al.* [16]. Ils utilisent *Functional Event Language* (FEL) comme un langage formel pour définir les souscriptions. FEL est un langage déclaratif et fortement typé (voir [81] pour plus de détails).
L'architecture du framework est fondée sur la DHT Chord. Pour la décentralisation de la gestion d'évènements, chaque nœud DHT peut être un détecteur d'évènements atomiques ou un détecteur d'évènements composites selon son ID. Le hachage permet au service de détection d'être placé sur un nœud DHT particulier. La Figure 4.7 illustre l'architecture du framework fondé sur FEL. Elle montre la distribution des services sur une DHT de 256 nœuds en montrant les 4 nœuds d'IDs : 14, 46, 97, 167. Le nœud 14 est un conteneur de deux services : Détecteur d'Evènements Composites (CED) et Détecteur d'Evènements Atomiques (AED). Les deux types d'évènements composite et atomique sont hachés pour associer les valeurs de CED et AED (10 et 191 respectivement) au nœud de la DHT d'ID 14 qui est le

successeur le plus proche des identifiants des services comme le montre la même figure. Ceci permet d'équilibrer automatiquement la charge entre les nœuds du réseau. Si un nouveau nœud rejoint la DHT avec un ID 255, alors l'AED d'ID 191 migre du nœud 14 au nœud 255.

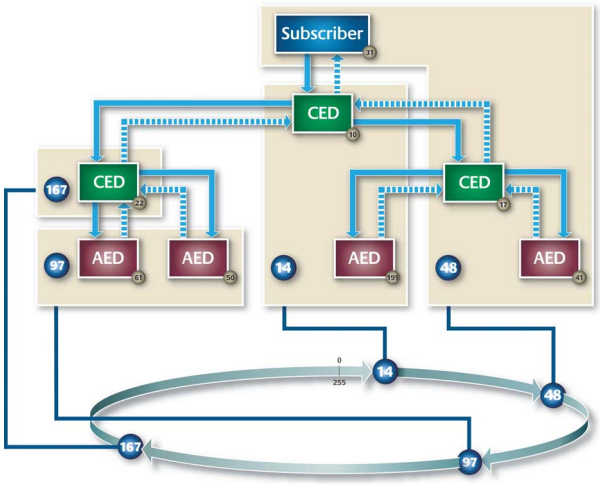


Figure 4.7 – FEL appliqué à Chord pour la gestion distribuée d'évènements [16]

Nous déduisons alors que ce système a résolu la difficulté de mise à l'échelle par l'utilisation de la DHT mais son problème majeur reste l'expressivité. En effet, le langage FEL utilisé ne permet que de spécifier le type de l'évènement souhaité (filtrage basé type). De plus, FEL permet la détection des patterns d'occurrence d'une suite d'évènements selon leurs types. Autrement dit, il permet de vérifier des compositions d'évènements telles que : l'évènement de type T_2 se produit après l'occurrence de l'évènement de type T_1.

Le dernier travail qui traite la composition d'évènements sur un système publier/souscrire basé DHT est celui de J. Qian *et al.* [82]. Les auteurs proposent d'utiliser deux structures sur chaque nœuds DHT : *Primitive Subscription Map* (PSM) pour la mémorisation des souscriptions primitives et *Composite Subscription Map* (CSM) pour la mémorisation des souscriptions composites.

Le défaut majeur de cette proposition est que tous les évènements primitifs doivent être vérifiés avant la vérification partielle des évènements sous-composites. Ceci s'avère inutile dans plusieurs exemples de compositions comme celles utilisant les opérateurs logiques *AND* et *XOR*.

4.4.4 Synthèse

Le Tableau 4.1 récapitule les approches étudiées dans les sections précédentes.

Tableau 4.1 – Traitement de la composition d'évènements dans les systèmes publier/souscrire

Approche	**Architecture**	**Technique**	**Relation de composition**	**Mise à jour**
RUBCES [12]	centralisée	à base de règles	relations logiques (AND/OR)	automatique selon les intérêts
P.R. Pietzuch *et al.* [13]	distribuée (selon les sources des CE)	FSA	contraintes temporelles	par le programmeur du système
M. Gao *et al.* [14]	centralisée	capteurs	contraintes spatio-temporelles	par programmation des capteurs
PSware [15]	centralisée	capteurs	opérateurs temporels (séquentiel et parallèle) et opérateurs logiques (conjonction, disjonction et négation)	par programmation des capteurs
Courtenage *et al.* [16]	DHT	FEL	occurrence d'une suite d'évènements selon leurs types	automatique selon les intérêts
JtangCSPS [82]	DHT	PSM/CSM	relations logiques et temporelles	par le programmeur du système

Les travaux présentés par [12], [14] et [15] adoptent une architecture centralisée et traitent généralement peu de relations de composition. Seul [12] permet la mise à jour automatique, pour l'intégration de nouvelles compositions, afin d'exprimer les nouveaux intérêts composites des utilisateurs. Le reste des approches centralisées ([14] et [15]) ne permet l'intégration de nouvelles compositions que par l'intervention d'une tierce partie (programmeur du système ou programmeur des capteurs).

Le travail présenté par [13] propose de distribuer les détecteurs des évènements composites selon leurs sources pour améliorer la scalabilité. Cependant, le problème d'ajout de nouvelles souscriptions composites reste restreint à l'intervention du programmeur du système. De plus, cette proposition ne traite que les contraintes temporelles pour la composition d'évènements.

Les travaux présentés par [16] et [82] traitent la composition sur une architecture P2P structurée. Avec le travail de [16], l'ajout de nouveaux patterns de souscriptions selon les

intérêts des utilisateurs est possible sans intervention du programmeur, mais il ne permet d'exprimer que le besoin d'occurrence d'une suite d'évènements selon leurs types (manque d'expressivité). En contrepartie, [82] permet de bien exprimer une variété de relations de compositions (temporelle et logique). Toutefois, l'intégration de nouvelle composition n'est possible que par l'intervention du programmeur du système.

4.5 CECube : filtrage des évènements composites dans MULUS

Après le traitement de la sémantique sur MULUS, nous constatons que la composition des données selon des relations logiques et temporelles est parmi les fonctionnalités cruciales de MULUS. Cette fonctionnalité permet de mieux répondre aux besoins des utilisateurs, d'améliorer le degré d'expressivité du système et d'alléger le trafic sur le réseau. Nous notons que l'intégration de nouveaux intérêts composites, sans intervention d'une tierce partie, est une propriété très importante pour notre plateforme.
Dans cette section, nous commençons par définir les relations de compositions traitées. Ensuite, nous détaillons l'approche proposée pour la composition d'évènements.

4.5.1 Relations de compositions des évènements

La composition d'évènements enrichit l'expression des souscriptions pour les utilisateurs et permet aussi une sélection d'évènements selon les contraintes de la vie quotidienne. Ainsi, l'intérêt d'un utilisateur peut être représenté par un évènement simple ou composite quand il y a une dépendance et relativité entre les évènements primitifs. Nous identifions les relations logiques et temporelles possibles pour la composition d'évènements. Pour ceci, nous traitons des relations logiques et temporelles riches et variées pour pouvoir couvrir tous les besoins.
Les relations logiques qui relient les évènements primitifs ou sous-composites peuvent être :

- la conjonction (AND) : si l'utilisateur demande deux évènements E_1 et E_2 ;
- la disjonction (OR) : si l'utilisateur demande l'évènement E_1 ou l'évènement E_2 ;
- le Ou-exclusif (XOR) : si l'utilisateur demande l'un des évènements E_1 ou E_2.

Les relations temporelles qui relient les évènements primitifs ou sous-composites sont :

- E_1 finit avant E_2 : l'utilisateur demande de recevoir E_1 puis E_2 après l'achèvement de

E_1 ;

- E_1 rencontre E_2 : l'utilisateur demande de recevoir E_1 et E_2 avec un temps d'intersection entre les deux ;
- E_1 chevauche avec E_2 : c'est équivalent à E_1 rencontre E_2 mais l'utilisateur cherche à commencer par E_1 et finir par E_2 ;
- E_1 finit avec E_2 : l'utilisateur demande de recevoir E_1 et E_2 et ces deux évènements finissent en même temps ;
- E_1 inclut E_2 : l'utilisateur demande de recevoir E_2 au moment de la réception de E_1 ;
- E_1 débute avec E_2 : E_1 et E_2 commencent au même instant ;
- E_1 est parallèle à E_2 : E_1 et E_2 occurrent simultanément.

4.5.2 Modélisation et indexation des évènements composites

Notre idée consiste à détecter l'évènement composite, tout en vérifiant ses relations logiques et temporelles, au fur et à mesure de la détection de ses évènements primitifs ou sous-composites. Nous rappelons que la détection des évènements primitifs se fait par les HBE en se basant sur l'ontologie de domaine comme expliqué dans le Chapitre 3. Alors, après la détection des évènements primitifs moyennant les nombres premiers, nous passons au traitement de la composition d'évènements.

Nous cherchons donc à mapper un évènement composite sur les nœuds DHT. Pour ce faire, nous proposons d'utiliser la structure d'un cube (*smart cube*) et d'un plan pour l'indexation des évènements composites et la vérification des relations de compositions. Ces structures sont créées et distribuées sur les nœuds DHT selon les souscriptions demandées par les utilisateurs.

Dans les prochaines sections, nous détaillons le modèle représentatif suggéré d'un évènement composite ainsi que les structures d'indexation proposées et leur utilisation durant la dissémination des évènements composites.

4.5.2.1 Modèle représentatif d'un évènement composite

Dans notre approche, nous définissons un évènement composite par l'agrégation d'évènements primitifs ou sous-composites avec un ensemble d'opérateurs logiques et des contraintes temporelles.

Nous notons l'univers d'évènement par CE, formé par l'ensemble d'évènements E. E peut être primitif ou sous-composite ou composite. Nous définissons un évènement composite par la formule 4.1.

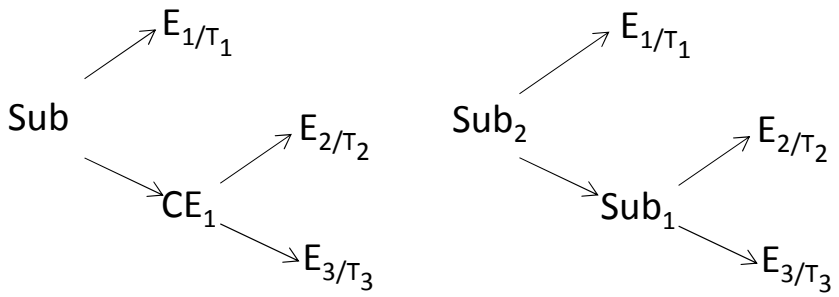
$$CE = [op_i(E_{i_{/Ti}}, E_{j_{/Tj}})] \tag{4.1}$$

Avec :

- E_i / E_j : évènement primitif ou sous-composite ;
- T_i / T_j : contrainte temporelle de l'évènement E_i / E_j respectivement ;
- op_i : relation logique entre E_i et E_j.

Si E_i est un évènement sous-composite, alors il est décomposé par la même formule jusqu'à obtenir les évènements primitifs.

La décomposition d'un évènement composite suit alors la structure d'un arbre présenté dans la Figure 4.8(a). Cette structure d'arbre est formée pendant la phase de souscription des parents aux fils et est suivie inversement (des fils aux parents) pendant la phase de publication. La composition d'évènements est alors vérifiée progressivement pour éviter la dissémination d'évènements inutiles.



(a) Décomposition sans intersection

(b) Décomposition avec des évènements communs

Figure 4.8 – Décomposition d'évènements composites sous forme d'arbre

Notre processus de composition détecte la répétition des évènements primitifs, sous-composites ou composites sur les nœuds DHT (Figure 4.8(b)). En effet, le hachage d'un évènement composite est obtenu par le hachage de la concaténation de ses éléments en excluant les opérateurs logiques et temporels entre eux afin de les collecter sur le même nœud DHT et vérifier par la suite leurs compositions puisqu'il y a des relations que nous

pouvons vérifier simultanément comme le *OR* et le *XOR*. Par conséquent, nous évitons de transférer les évènements ne répondant pas aux besoins des utilisateurs par la vérification de chaque composition aux nœuds intermédiaires avant de continuer leur transfert vers les utilisateurs.

4.5.2.2 Indexation des évènements atomiques et composites

Dans la suite, nous détaillons les deux méthodes d'indexation proposées pour la mémorisation et le matching des évènements sur les nœuds DHT. Nous rappelons que nous avons élaboré l'indexation sémantique des évènements dans le chapitre précédent. Nous utilisons alors les identifiants sémantiques des évènements atomiques déjà définis. Dans la suite de ce rapport, l'expression "identifiant d'un évènement" signifie "identifiant sémantique d'un évènement" pour les évènements atomiques.

1. Structure de Cube

Pour garantir un matching efficace sur un service d'évènements distribué, nous proposons un espace d'indexation à trois-dimensions que nous appelons CECube. CECube est un cube intelligent maintenu par chaque serveur d'évènements (nœud DHT) responsable d'une souscription composite. Il est utilisé pour mémoriser les différents modèles de compositions proposés par les utilisateurs. Il maintient tous les évènements membres qui composent un évènement composite, avec les contraintes temporelles et les opérateurs logiques. L'architecture du cube est montrée par la Figure 4.9.

Le premier axe du CECube (*I(CE)*) représente le hachage des identifiants des évènements composites obtenu par le hachage de la concaténation des identifiants sémantiques des évènements primitifs qui le composent. Le deuxième axe (*I(E)*) représente le hachage de l'identifiant des évènements primitifs (identifiants sémantiques) ou des évènements sous-composites. Quant au troisième axe (*I(T)*), il représente l'instant d'occurrence des évènements.

Pour le hachage des identifiants, nous utilisons la fonction de hachage uniforme *sha-1* qui donne un hachage sans conflits. Nous rappelons aussi que le hachage des évènements composites est fait sans maintenir les relations logiques/temporelles entre ces membres. Ceci permet de réduire aussi le nombre de CECubes construits pour l'indexation des évènements composites sur les nœuds DHT.

Chaque point de ce cube maintient un ensemble de valeurs nommées *cubeValueSet* et noté *C* sur les figures. Le *cubeValueSet* contient :

– un bit indicateur (*flag bit*) : prend la valeur 0 ou 1 et est nommé *eventFlag* pour indiquer si une souscription composite contient un évènement E à un instant T. Il permet alors de vérifier l'appartenance d'un évènement atomique à une composition ;
– un vecteur d'identifiant des souscrits : est noté *SIV*. Il mémorise les identifiants de tous les nœuds souscrits à l'évènement correspondant ;
– les relations logiques/temporelles avec les autres évènements de la même composition.

Comme montré par la Figure 4.9, un plan perpendiculaire à l'axe *I(CE)* est identifié par *[I(CE), *, *]* et il représente la composition de l'évènement *CE*. De même, une droite parallèle à l'axe *I(CE)*, identifiée par *[I(CE), *, I(T)]*, forme la composition de l'évènemet CE à un instant T donné. Ainsi, les cases mises à 1 de cette droite déterminent les évènements primitifs appartenant à ce même évènement composite.

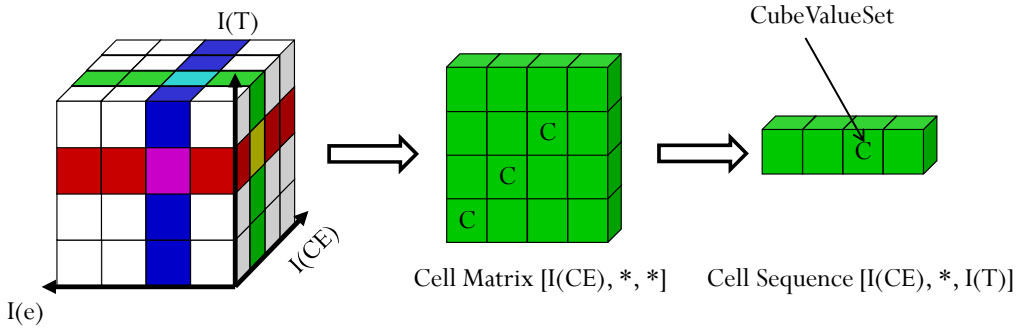


Figure 4.9 – Indexation d'un évènement composite sur CECube

CECube sera créé au niveau des nœuds responsables des évènements composites et redirigera le routage des souscriptions vers les nœuds responsables des évènements élémentaires. Il permet le filtrage des évènements par une simple recherche binaire effectuée sur ce cube. L'utilisation de cette structure sera détaillée prochainement dans les phases de souscription et de publication.

2. Structure du vecteur de matching

Pour vérifier la composition totale de l'évènement composite à travers la structure de CECube, nous définissons un vecteur de matching nommé *EventMatchingVector*. La structure d'*EventMatchingVector* est utilisée en association avec CECube pour la détection des évènements composites. Elle est montrée par la Figure 4.10.

L'*EventMatchingVector* mémorise les sommations des valeurs d'*eventFlag* pour chaque évè-

nement composite du cube. C'est-à-dire, chaque valeur de ce vecteur représente le résultat de sommation des *eventFlags* du plan *[I(CE), *, *]* correspondant. Chaque élément du vecteur est mis à 0 quand tous les membres (primitif ou composite) de l'évènement composite correspondant sont vérifiés. Si toutes les valeurs de ce vecteur sont mises à 0, un évènement de notification est envoyé aux souscrits mémorisés par *SIV*. Autrement dit, un évènement de notification est déclenché quand une opération de *OR* logique appliquée à ce vecteur retourne la valeur 0. En conséquence, cette structure aide et accélère la détection des évènements composites dont les contraintes logiques/temporelles sont vérifiées à travers le CECube.

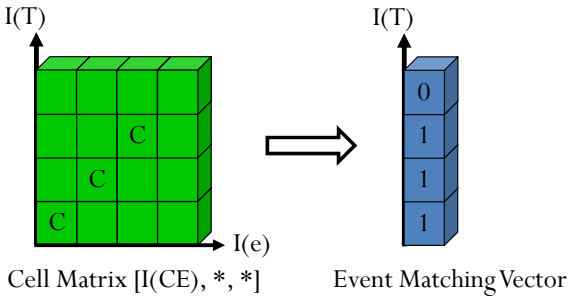


Figure 4.10 – Modélisation du vecteur de matching

3. Structure de plan pour indexation d'évènements primitifs

Pour alléger la mémorisation et la recherche des évènements primitifs sur un nœud DHT responsable, nous définissons la structure d'un plan défini par deux axes $I(E)$ et $I(T)$ comme montré par la Figure 4.11. $I(E)$ est l'indice de la valeur hachée de l'identifiant d'un évènement primitif (identifiant sémantique) et $I(T)$ est l'indice de l'instant d'occurrence de cet évènement. Cette structure est maintenue alors par les GTDM des différents HBE.

Chaque point de ce plan est représenté par une cellule contenant un ensemble de valeurs nommé *planvalue* qui englobe deux valeurs : la première valeur est un bit indicateur *eventFlag* qui indique qu'une souscription primitive contient l'évènement E à l'instant T. L'*eventFlag* est mis à 1 pour indiquer qu'il y a une souscription qui demande l'évènement primitif E_i à l'instant T_i. La deuxième valeur est la liste des identifiants des souscrits à cet évènement qu'on note *SIV* (*Subscriber Identifier Vector*). Les souscrits peuvent être un point d'accès d'un utilisateur qui est souscrit à un évènement primitif ou un nœud DHT qui est responsable d'une composition d'évènements. En effet, un évènement primitif peut répondre à une souscription simple comme il peut appartenir à un évènement composite, donc il doit se réunir

avec les autres évènements sur un nœud possédant la structure CECube définie auparavant. Le *SIV* permet alors de déterminer le chemin de dissémination des évènements primitifs.

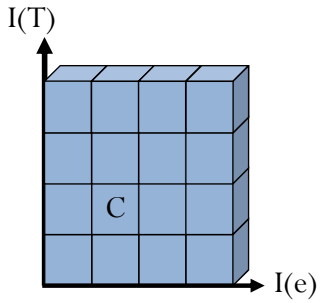


Figure 4.11 – Structure de plan pour l'indexation d'un évènement primitif

4.5.3 Démarche proposée pour le matching des évènements composites

Dans cette section, nous détaillons la démarche à suivre pendant la phase de souscription et la phase de publication en utilisant les structures définies dans la section précédente.

4.5.3.1 Phase de souscription

Nous définissons l'Algorithme 1 pour la phase de souscription. Nous traitons les deux cas possibles : souscription simple et souscription composite.

En recevant une souscription primitive d'un utilisateur, le processus commence par la recherche de l'identifiant sémantique de cette souscription comme expliqué dans le Chapitre 3. Une fois l'identifiant déterminé, il est utilisé par la suite pour son routage par le protocole de la DHT vers le nœud responsable. En arrivant à ce nœud, nous finissons par la mise à jour de la valeur correspondante dans le plan en ajoutant le nœud expéditeur dans *SIV* et en mettant l'*eventFlag* à 1.

En recevant une souscription composite d'un utilisateur selon le pattern présenté par la formule 4.1, nous utilisons CECube pour l'indexer. Le processus commence par le hachage de la souscription après avoir éliminé tous les opérateurs et trié les évènements par ordre alphabétique (chaque évènement primitif est représenté par son identifiant sémantique). Cette étape nous permet de vérifier, sur le même nœud DHT, des évènements composites formés par les mêmes évènements mais avec des relations logiques/temporelles différentes. Ensuite,

Algorithme 1 : processus de souscription

```
   Data : subscription Sub, sender S
   Result : E1, E2, op
 1 begin
 2     switch type of Sub do
 3         case Composite do
 4             if Sub already mapped then
 5                 updateSIV(S);
 6             else
 7                 decompose(Sub, E1, E2, op);
 8                 cube(Sub, E1, E2, op, S);
 9                 updateEventMatchingVector(Sub, E1, E2);
10                 send(E1);
11                 send(E2);
12             end
13             break;
14         end
15         case Primitive do
16             plan(Sub, S);
17             break;
18         end
19     end
20 end
```

nous envoyons la souscription par le protocole DHT vers le nœud responsable (*root*). En arrivant à ce nœud, nous vérifions tout d'abord si cette souscription est indexée auparavant, telle quelle ou avec les mêmes composants mais avec d'autres relations logiques/temporelles. Si c'est le cas, le processus d'indexation se limite à l'ajout de l'identifiant du point d'accès à *SIV* et l'ajout des nouvelles relations en cas de besoin. Nous finissons, dans ce cas, la phase de souscription. Si la composition n'est pas encore indexée sur le *root*, alors nous utilisons la structure de CECube pour indexer l'évènement composite en mettant à jour la cellule correspondante comme expliqué dans la section 4.5.2.2. Selon le résultat de hachage, la valeur de *cubeValueSet* est mise à :

(eventFlag = 1, SIV = identifiant du souscrit, op_1 = relation logique, op_2 = relation temporelle).

Ensuite, nous mettons à jour l'*EventMatchingVector* par la sommation des *eventFlags*. Enfin, nous passons à la décomposition de cet évènement en deux évènements sous-composites et la relation entre eux et nous réitérons ce même processus. Ce processus est répété pour chaque

évènement sous-composite jusqu'à retrouver une souscription déjà faite ou atteindre tous les évènements primitifs. La Figure 4.12 montre le mapping d'un exemple de souscription composite présenté dans la Figure 4.8(a). Pour cette souscription, deux CECubes et deux *EventMatchingVectors* sont créés sur les nœuds N_0 et N qui sont responsables des évènements composites et trois plans sont créés sur les nœuds N_1, N_2 et N_3 responsables des évènements primitifs.

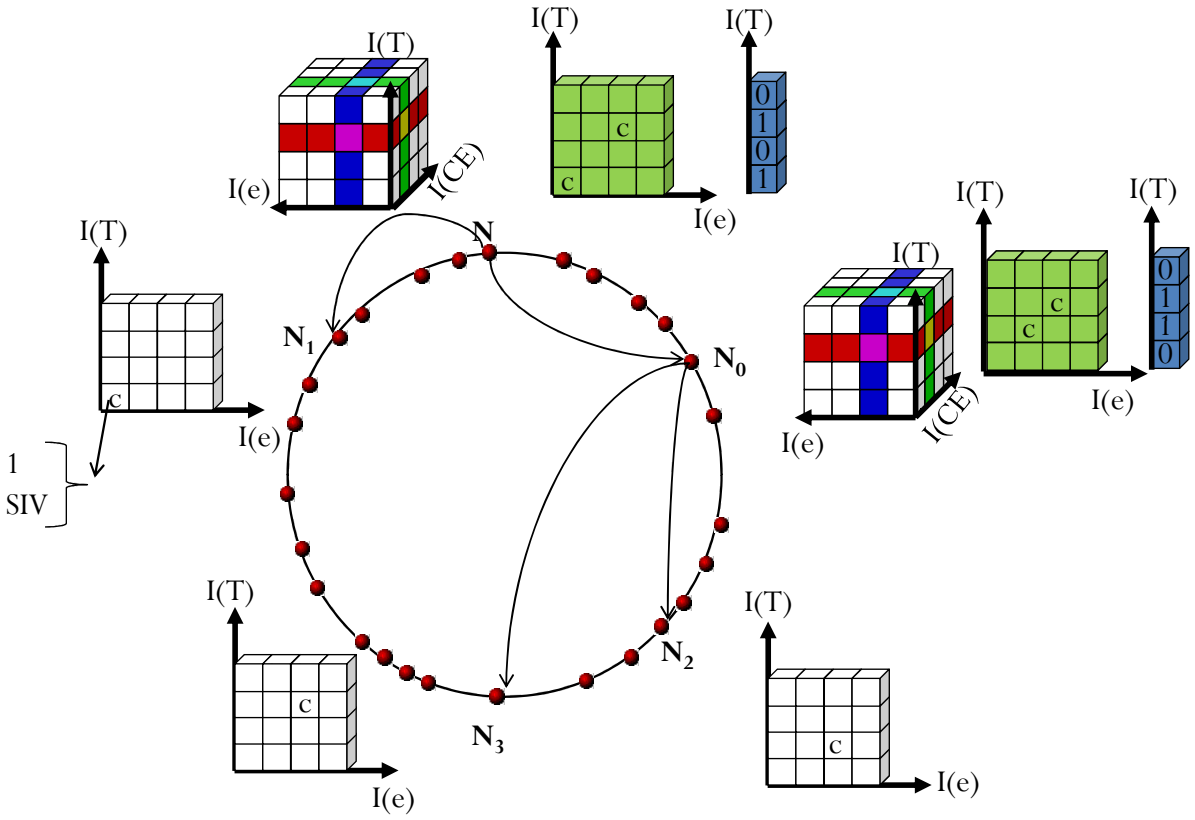


Figure 4.12 – Exemple d'application du processus de souscription composite

4.5.3.2 Phase de publication

Pendant la phase de publication, nous cherchons à collecter les évènements des différents nœuds DHT au fur et à mesure de leur réception pour satisfaire les souscriptions composites. Le processus d'agrégation respecte l'arbre de décomposition (présenté précédemment dans la Figure 4.8) en allant des fils (évènements primitifs) vers le père (évènement composite) tout en vérifiant les relations de compositions. Ceci permet d'éliminer les transferts des évènements qui ne respectent pas le modèle de composition.

L'Algorithme 2 décrit les étapes de la phase de publication.

Algorithme 2 : processus de publication

```
Data : publication E, receiver R
Result : E1, E2, op
1  begin
2      switch type of R do
3          case N_of_Primitive_Event do
4              SIV ⟵ scanPlan(e);
5              for dest ∈ SIV do
6                  send(E, dest);
7              end
8              break;
9          end
10         case N_of_Composite_Event do
11             CE ⟵ scanPlan([∗, I(e), ∗]);
12             for ce ∈ CE do
13                 if scanCubeValue([I(ce), *, *]) then
14                     updateEventMatchingVector(ce);
15                     if (EventMatchingVector(ce)==0) then
16                         SIV ⟵ scanPlan(ce);
17                         for dest ∈ SIV do
18                             send(ce, dest);
19                         end
20                     end
21                 end
22             end
23             break;
24         end
25     end
26 end
```

Quand une publication est reçue par le nœud racine (par indexation sémantique), ce dernier vérifie si elle relie une souscription reçue auparavant. Il utilise alors la structure du plan créé pendant la phase de souscription. Pour ceci, il consulte la cellule de coordonnée $(I(E), I(T))$. Si son *eventFlag* est mis à "0", alors il n'y a pas de souscrits à cet évènement. Dans le cas où l'*eventFlag* est mis à "1", il consulte le *SIV* pour déterminer les identifiants des souscrits destinataires de cette publication. Ces derniers peuvent être soit des utilisateurs soit d'autres nœuds DHT responsables de la composition des évènements. Lorsqu'une publication est envoyée à un nœud DHT, ce dernier regarde son CECube créé pendant la phase de

souscription. La recherche est concentrée donc sur le plan $[*, I(E), *]$ perpendiculaire à l'axe $I(E)$ pour vérifier si cette publication appartient à des souscriptions composites en regardant les *eventFlags*. Lorsqu'un *eventFlag* est mis à "1", nous regardons le plan $[I(CE), *, *]$ correspondant et nous analysons tous les *cubeValueSets* pour vérifier le matching et les relations logiques/temporelles. Une fois qu'une composition est vérifiée, la valeur de *EventMatchingVector* est mise à jour. Quand toutes les valeurs de *EventMatchingVector* sont mises à "0", une notification est envoyée aux souscrits en utilisant le *SIV* stocké dans le *cubeValueSet*. La phase de publication se termine une fois toutes les notifications arrivent aux utilisateurs souscrits.

Nous revenons sur l'exemple traité à la phase de souscription faite dans la section 4.5.3.1. Nous supposons que notre plateforme reçoit les 3 publications attendues selon la souscription composite émise. Ces publications sont alors envoyées aux nœuds N_1, N_2 et N_3. En utilisant les plans d'indexation, ces derniers les envoient aux nœuds responsables des compositions qui sont N_0 et N. N_0 à son tour envoie la composition obtenue au nœud N après vérifications des relations logiques et temporelles faites sur CECube. Le nœud N est alors responsable de notifier l'utilisateur final.

4.6 Évaluation des performances

Pour évaluer notre approche, nous continuons l'implémentation des structures proposées sur le système publier/souscrire basé DHT Scribe [20]. Nous ne traitons pas dans cette partie la dynamicité des nœuds DHT. L'expérimentation est faite sur le simulateur FreePastry en utilisant une machine Linux dotée d'un processeur i5 CPU 2.53 GHz et 4 Go de RAM. Nous montrons dans une première partie les avantages de notre approche en mesurant le temps de routage et le nombre de sauts dans différents cas. Puis, nous comparons le temps de routage par rapport à JTangCSPS [82] pour vérifier les performances de notre système.

Les métriques définies pour l'expérimentation sont :

- le délai de routage : il inclut le temps mis pour l'indexation, le temps mis pour la décomposition et le temps d'acheminement depuis l'utilisateur souscrit jusqu'à le nœud root pour une souscription, ou du producteur au consommateur pour une publication.
- le nombre de sauts : pour une souscription, c'est le nombre de sauts effectués pour passer une souscription d'un utilisateur souscrit au nœud root. Pour une publication, c'est le

nombre de sauts effectués pour passer une publication du producteur au consommateur.

4.6.1 Réduction de trafic grâce à la composition d'évènements

Nous rappelons que notre but est de répondre aux besoins composites des utilisateurs sur les RSD. Pour atteindre notre objectif, nous avons suggéré d'intégrer un système publier/-souscrire assurant la composition d'évènements. Ce dernier permet de répondre aux besoins des utilisateurs et de réduire par conséquent le trafic sur le réseau en évitant les transferts inutiles entre les utilisateurs.

Pour évaluer notre proposition, nous avons testé deux scénarios.
Dans le premier scénario, qui a fait l'objet du chapitre précèdent, nous avons injecté 1000 souscriptions atomiques. Il a permis de comparer un RSD fondé sur un système publier/souscrire par rapport aux réseaux existants. Ce scénario a prouvé que le nombre de notifications envoyées aux utilisateurs est remarquablement réduit (environ 82%) en utilisant notre plateforme MULUS.

Dans le deuxième scénario, nous utilisons les mêmes 1000 souscriptions atomiques du premier scénario pour injecter 200 souscriptions composites, formée chacune de 5 souscriptions primitives. Nous produisons 4000 publications à envoyer sur le réseau pour chacun de ces deux scénarios. La comparaison entre ces scénarios permet de montrer l'apport de la composition d'évènements dans un RSD.

Nous avons mesuré le nombre de notifications reçues par les utilisateurs dans ces deux scénarios. Le résultat a montré que l'injection des souscriptions composites réduit davantage le trafic global. La réduction atteint environ 65% par rapport à l'utilisation d'un système publier/souscrire (premier scénario). Ces résultats sont montrés par les courbes de la Figure 4.13.

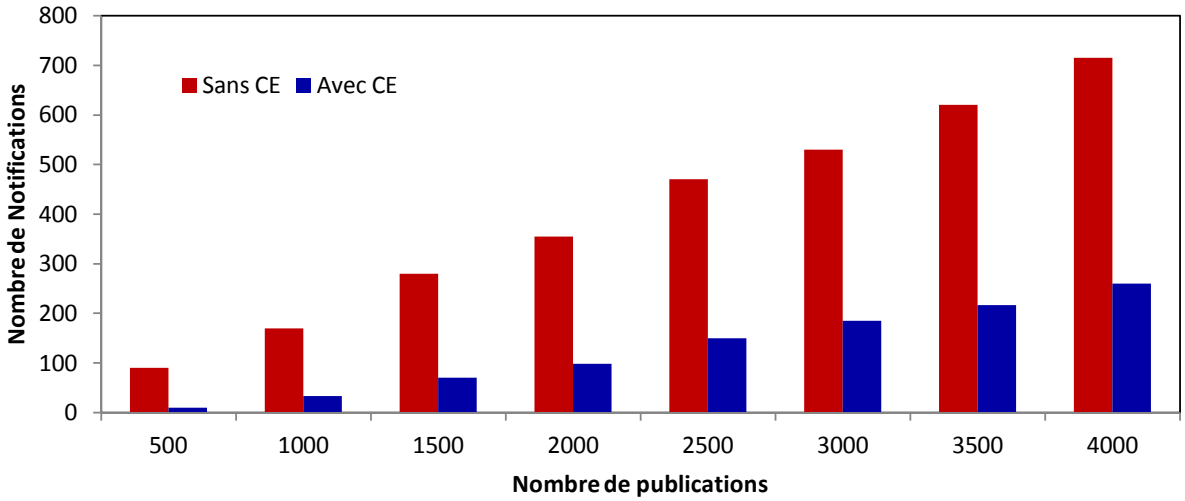


Figure 4.13 – Réduction du trafic par un système publier/souscrire et par composition d'évènements

4.6.2 Comparaison entre modèles de souscriptions composites : avec intersections et sans intersections

Nous remarquons qu'en pratique, il y a des intérêts communs entre les utilisateurs des RS. Le nombre de ces intérêts augmente s'il y a des données importantes ou des connaissances communes entre eux. Ceci est traduit par des intersections entre les évènements composites montrées auparavant par la Figure 4.8.
La structure de CECube que nous avons proposée facilite la détection de ces intersections et permet de faire le matching de plusieurs évènements simultanément.

Pour vérifier la performance de CECube, nous proposons de comparer le temps de routage pour un modèle de composition avec intersections et celui pour un modèle sans intersections. Ces deux modèles sont montrés par les Figures 4.15 et 4.14 respectivement. Les évènements composites montrés par la Figure 4.14 ne présentent pas d'intersections. Ils sont composés de 36 souscriptions primitives formant 10 souscriptions composites envoyées sur notre service d'évènements. Pour le deuxième modèle montré par la Figure 4.15, chaque souscription composite envoyée est réutilisée pour la composition d'une autre souscription composite. Nous avons ainsi utilisé 12 souscriptions primitives pour composer le même nombre de souscriptions composites que le premier modèle (10 souscriptions composites).

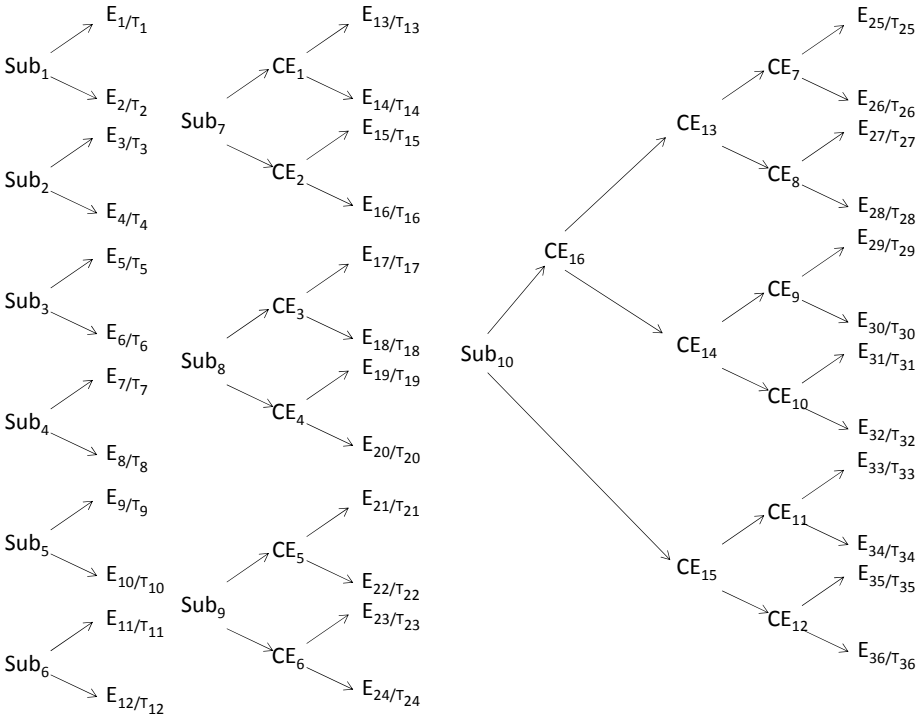


Figure 4.14 – Exemple de 10 souscriptions composites sans intersections

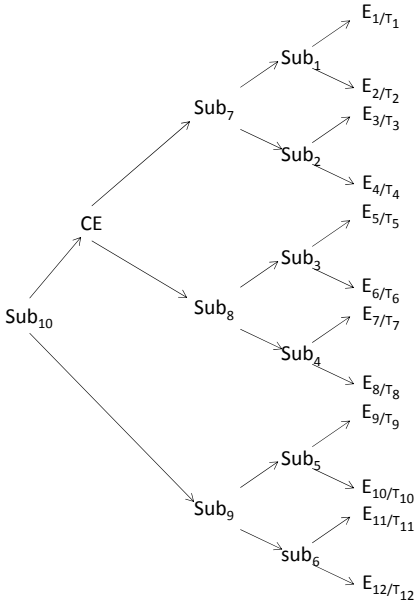


Figure 4.15 – Exemple de 10 souscriptions composites avec intersections

Les deux courbes de la Figure 4.16 montrent le temps de routage pour la phase de souscription avec des intersections et sans intersections. En comparant ces deux courbes, nous

remarquons que le temps de routage des souscriptions composites sans intersections entre elles est légèrement supérieur à celui des souscriptions avec intersections (répétitions). La différence est de 150 ms pour un exemple simple (celui montré par les deux figures précédentes) mais elle peut être plus importante dans la pratique, où plus d'intersections sont impliquées. En effet, le nombre de cubes (CECube) créés est réduit quand les compositions se répètent même avec des relations de compositions différentes. Nous rappelons aussi que le hachage est appliqué toujours sur l'ensemble des évènements primitifs sans inclure les relations de compositions. Par conséquent, quand des souscriptions primitives ou sous-composites sont répétées, même avec différentes relations, le matching des évènements composites ou sous-composites se fait dans le même cube. Le nombre de nœuds visités est aussi réduit puisqu'en retrouvant un nœud déjà responsable d'une composition, la souscription s'arrête sans finir sa décomposition et sans acheminer les souscriptions primitives.

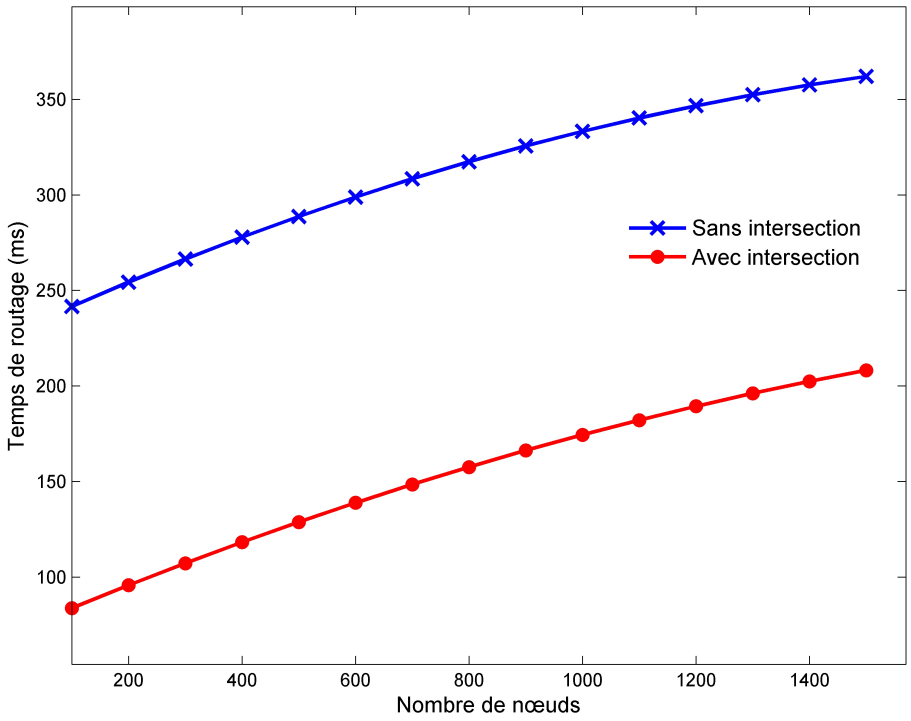


Figure 4.16 – Temps de routage en fonction du nombre de nœuds pendant la phase de souscription

Ce raisonnement est aussi vérifié par le test expérimental et il est montré par les courbes de la Figure 4.17. Ces courbes montrent le nombre de sauts pour acheminer des souscriptions composites avec et sans intersections. Le nombre de sauts est considérablement réduit avec

des souscriptions avec intersection présentées par le même exemple de modèle. La différence a été de 40 sauts pour notre exemple et elle peut être beaucoup plus importante dans la pratique en augmentant le nombre de nœuds DHT et en variant les intérêts des utilisateurs.

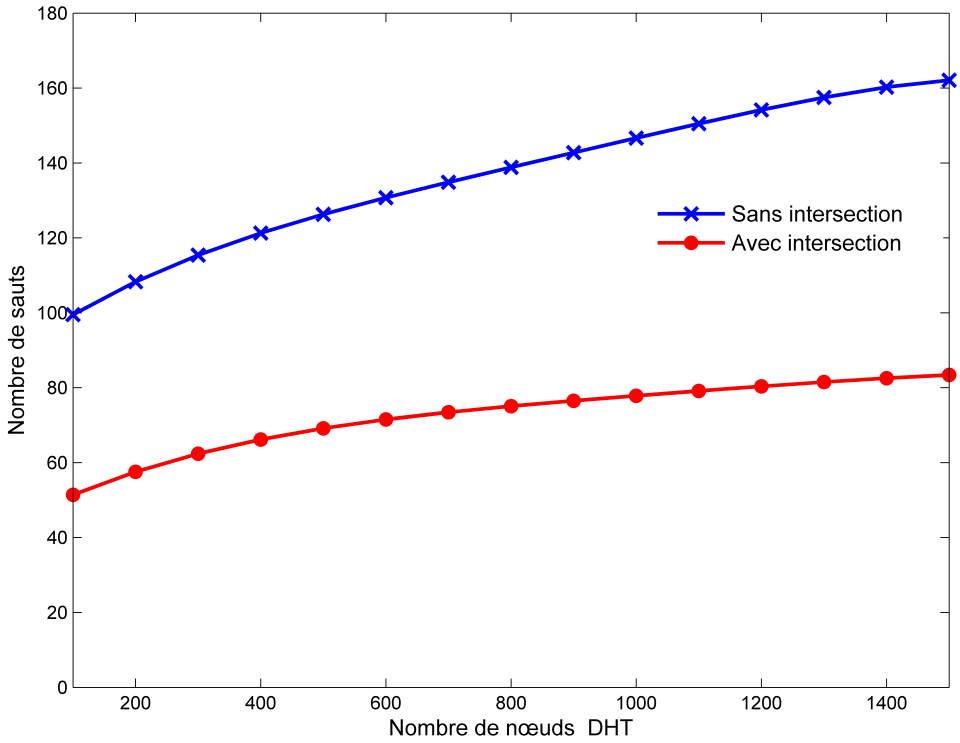


Figure 4.17 – Nombre de sauts en fonction du nombre de nœuds pendant la phase de souscription

Les deux courbes de la Figure 4.18 montrent le temps de routage pour la phase de publication sachant que les souscriptions soumises à notre plateforme sont de deux modèles : avec intersections et sans intersections, comme montré par l'exemple précédent (Figures 4.14 et 4.15). En comparant ces deux courbes, nous remarquons que le temps de routage des souscriptions composites sans intersections est légèrement supérieur à celui des souscriptions avec intersections (répétitions). La différence est de 500 ms pour cet exemple simple mais elle devra être plus importante dans un cas réel, où nous retrouvons plusieurs intérêts communs sur un réseau à large échelle. En effet, le matching et la vérification des relations logiques et temporelles sur un nombre de CECubes réduit est plus rapide et plus simple que si le nombre de CECubes et le nombre de roots sont plus grands. En fait, avec des souscriptions répétées, le nombre de nœuds visités est réduit puisque la souscription s'arrête en retrouvant

un nœud déjà responsable.

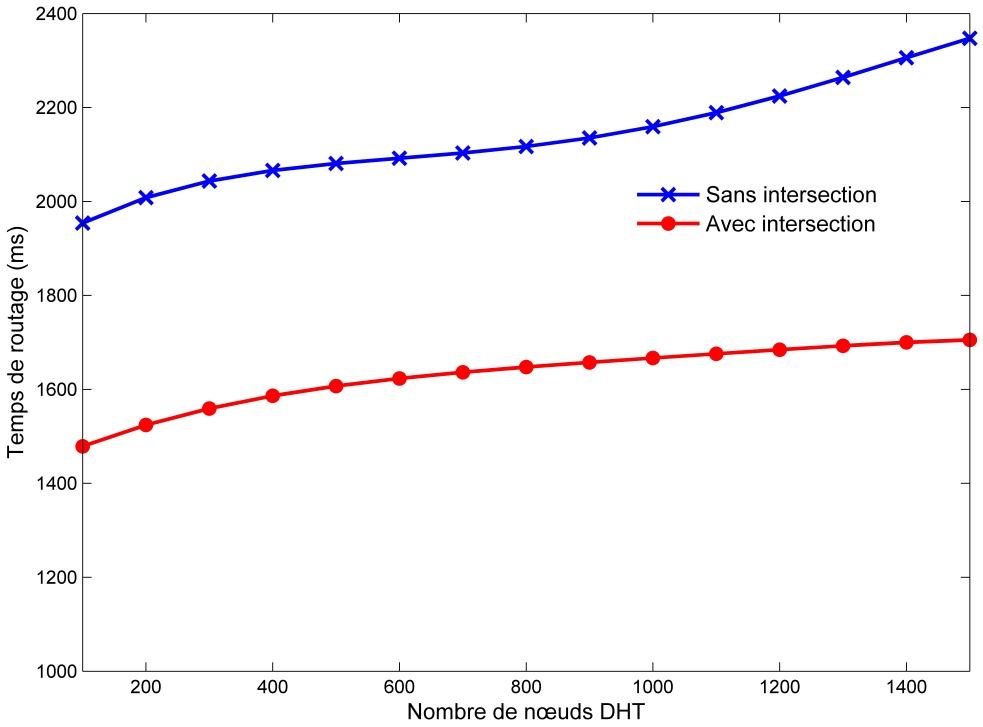


Figure 4.18 – Temps de routage en fonction du nombre de nœuds pendant la phase de publication

4.6.3 Étude de la scalabilité de MULUS

Pour prouver la performance de la topologie P2P structurée DHT et de la méthode d'indexation proposée, nous avons fait des tests en variant le nombre de nœuds DHT du service d'évènements. Les courbes des Figures 4.16 et 4.17, montrant le temps de routage, restent presque constantes en augmentant le nombre de nœuds DHT de notre réseau. Ceci prouve la scalabilité de notre plateforme basée DHT.

Ces courbes montrent aussi que le temps de routage d'une publication est plus important que le temps de routage d'une souscription. Ceci est vrai puisque nous vérifions le matching des évènements primitifs ainsi que les relations de compositions. Le temps de publication augmente légèrement en augmentant la taille de notre réseau mais il reste toujours acceptable, ce qui prouve la scalabilité de notre plateforme.

De même, la Figure 4.18 montre que le nombre de sauts pendant la phase de souscription augmente légèrement avec le nombre de nœuds déployés. Autrement dit, le nombre de

sauts est faiblement sensible au nombre de nœuds DHT formant le réseau. Ceci permet de juger aussi la scalabilité. Ainsi, notre plateforme assure un routage flexible et rapide grâce à l'architecture DHT et à la structure de CECube proposée.

4.6.4 Comparaison avec JTangCSPS

Pour évaluer la performance de notre plateforme, nous comparons aussi notre approche à JTangCSPS du point de vue temps de routage. Nous rappelons que le temps de routage pour CECube inclut le temps mis pour l'indexation, le temps mis pour la décomposition et le temps d'acheminement depuis l'utilisateur souscrit jusqu'au root pour une souscription ou du producteur au consommateur pour une publication. Pour JtangCSPS, le temps de routage inclut le temps d'acheminement des messages, le temps mis pour la construction de l'arbre de la souscription composite et leur division sur les différents nœuds.

Nous avons testé avec différents scénarios en variant le nombre de souscriptions primitives de 2 à 20. Chaque broker effectue 20/i souscriptions composites dont chacune présente i souscriptions primitives avec i=2, 5, 10 et 20. Pour simplifier la comparaison, nous utilisons seulement l'opérateur logique *AND* pour la composition.

Les courbes de la Figure 4.19 montrent que l'approche de CECube a un temps de routage largement inférieur à celui de JTAngCSPS. Ceci est justifié par le fait que l'indexation par CECube est moins coûteuse que la construction de l'arbre de JTangCSPS.
Ces courbes montrent aussi que le temps de routage se stabilise après 5 souscriptions primitives avec JTangCSPS et après 10 souscriptions primitives avec CECube. Ceci est non important dans la pratique, où nous trouvons un grand nombre de souscriptions traduisant les différents intérêts. Le plus intéressant est le temps de routage mis en augmentant le nombre de nœuds du réseau. Par ailleurs, le temps de routage avec CECube est presque le même avec 500 et 1000 nœuds alors qu'il augmente de 800 ms avec JTangCSPS entre 500 et 1000 nœuds. Ceci prouve encore la scalabilité de CECube déployé sur une topologie P2P structurée qui assure un routage flexible et rapide grâce à l'architecture DHT.

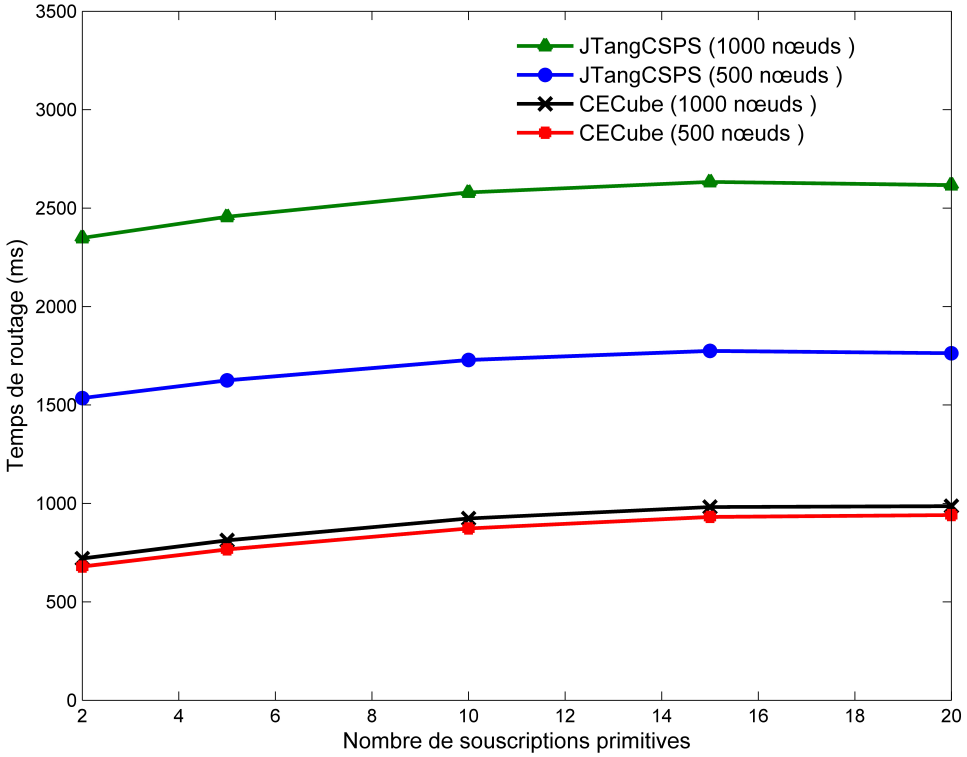


Figure 4.19 – Temps de routage en fonction du nombre de souscriptions primitives avec JTangCSPS et CECube

4.7 Conclusion

Dans ce chapitre, nous avons traité la composition d'évènements pour mieux répondre aux besoins composites des utilisateurs, réduire le trafic et améliorer la scalabilité sur notre plateforme MULUS. Pour ceci, nous avons proposé deux structures d'indexation distribuées sur le service d'évènements basé DHT. La première structure est le CECube qui permet d'indexer un évènement composite sur trois dimensions, en utilisant un bit d'indication. Cette méthode permet d'enregistrer les évènements composites facilement et de vérifier une telle composition dans un temps acceptable. La deuxième structure est un plan pour mémoriser les évènements primitifs sur les nœuds responsables en utilisant aussi un bit d'indication. Ceci permet de vérifier l'occurrence d'un évènement à un instant bien déterminé.

Avec les résultats expérimentaux, nous avons montré que notre approche réduit considérablement le trafic par rapport aux systèmes publier/souscrire ne traitant pas la composition d'évènements. Elle réduit aussi le temps de routage en la comparant au système JtangCSPS.

Les résultats expérimentaux prouvent aussi la scalabilité de notre approche en sachant que le temps de routage reste presque constant avec une augmentation de la taille du réseau. De plus, notre plateforme permet un matching exact entre les préférences de l'utilisateur et les données partagées par la sémantique et la composition d'évènements.

Nous rappelons que MULUS utilise les équipements des utilisateurs comme serveurs d'évènements pour éviter le recours à des serveurs externes et les frais imposés par les fournisseurs de services. Pour améliorer la qualité de service de notre plateforme MULUS, il est indispensable de traiter la disponibilité de données surtout que les équipements utilisés se caractérisent généralement par un dynamisme fort. Dans le chapitre suivant, nous proposons une nouvelle approche pour assurer la disponibilité des données et une livraison sans perte aux utilisateurs de MULUS.

Chapitre

5 Disponibilité des contenus dans MULUS

5.1 Introduction

Comme détaillé dans le deuxième chapitre, nous avons proposé d'utiliser une topologie P2P structurée pour notre plateforme MULUS pour améliorer ses performances. La caractéristique la plus marquante de cette topologie est que la communication entre les utilisateurs s'effectue directement sans faire intervenir des serveurs centraux en utilisant les dispositifs des utilisateurs comme serveurs/gateways. Ceci nous permet de surmonter les contraintes et les frais exigés par les fournisseurs de services des RS. Les réseaux P2P présentent aussi une plateforme efficace et scalable pour les applications distribuées en particulier les RS qui soutiennent des millions d'utilisateurs en ligne.

Contrairement à d'autres systèmes distribués où les défaillances peuvent être considérées comme rares ou anormales, les réseaux P2P de ce type restent constamment dans l'état de *churn* (forte dynamicité). Ceci est dû au comportement dynamique de ces systèmes dans lesquels l'arrivée/départ des pairs participants sont très fréquents et désynchronisés. Ce comportement influence la disponibilité des données stockées sur les pairs et la livraison de données (cas de vidéo streaming par exemple).

Le problème de disponibilité de données sur les RSD est un sujet d'actualité où peu de solutions sont proposées. En effet, la majorité des RSD proposés s'intéresse plus à la confidentialité des données qu'à leur disponibilité. En outre, les utilisateurs des RSD ont la liberté de stocker leurs contenus sur des équipements de leurs choix. En réalité, ces équipements n'ont pas nécessairement une disponibilité élevée et ils peuvent être même défaillants. Par conséquent, assurer 24/7 disponibilité des contenus des utilisateurs stockés dans un cadre décentralisé est un défi.

Ce chapitre aborde donc la dernière problématique de cette thèse qui est la disponibilité de

données dans les RSD. Nous commençons par la représentation des concepts de base. Ensuite, nous étudions les solutions proposées pour résoudre cette problématique sur : les RSD, les systèmes publier/souscrire basés DHT et les réseaux P2P structurés. Enfin, nous proposons deux stratégies de réplication pour traiter la disponibilité de données sur MULUS et nous présentons les résultats expérimentaux pour montrer l'efficacité de ces deux stratégies.

5.2 Concepts de base

5.2.1 Placement de données sur DHT

Dans une DHT, un identifiant, dit aussi clé, est affecté à chaque pair et à chaque bloc de données. L'identifiant du bloc de données est généralement le résultat d'une fonction de hachage exécutée sur le bloc. Tous les identifiants sont arrangés selon une structure logique qui peut être représentée par un anneau comme dans Chord [46] et Pastry [46] ou un tore de dimension d comme dans CAN [63] et Tapestry [49]. Le pair dont l'identifiant est le plus proche numériquement de la clé du bloc est appelé la racine du bloc ou nœud responsable de la clé. Il est donc responsable de maintenir ce bloc de données. Chaque DHT définit sa méthode de routage appropriée mais le principe du nœud root reste le même.

5.2.2 Définition de churn

Comme illustré dans la Figure 5.1, le temps de session est la durée entre le moment où un nœud rejoint un réseau jusqu'à ce qu'il quitte le réseau par la suite. En revanche, la durée de vie d'un nœud est le temps entre le moment où il arrive sur le réseau pour la première fois jusqu'au moment où il le quitte en permanence [83]. Enfin, la disponibilité d'un nœud est souvent définie comme la somme de ses temps de session divisée par sa durée de vie.

En pratique, l'arrivée et le départ non anticipés des pairs causent une désorganisation de la structure du réseau et par suite une mauvaise gestion de données. Ceci influence beaucoup la performance et la disponibilité des systèmes à base de DHT si l'arrivée et le départ de chaque nœud deviennent très fréquents et de durées très courtes et variées. Ce phénomène est appelé *churn*.

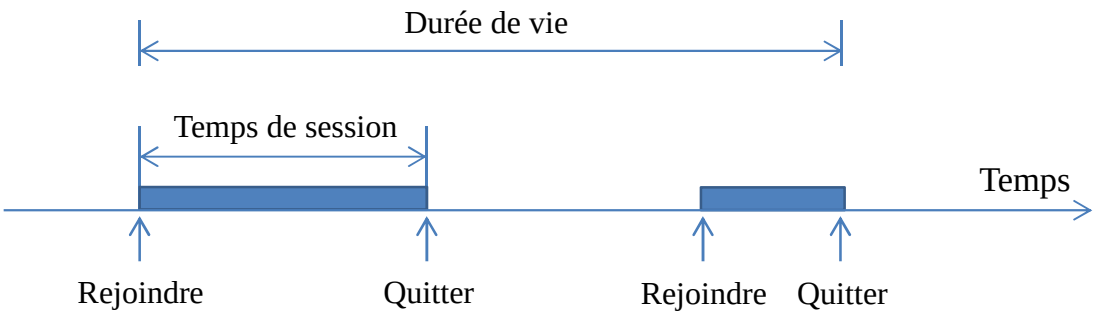


Figure 5.1 – Métriques de churn

5.2.3 Problème de churn

Les systèmes basés DHT utilisent une fonction d'indexation pour localiser les données sur les pairs connectés au réseau. Un système publier/souscrire utilisant DHT pour le routage des évènements (souscriptions/publications) nécessite une grande disponibilité des nœuds DHT. En particulier, les serveurs de notre plateforme MULUS sont les terminaux des utilisateurs alors le taux de churn est très élevé.

5.2.4 Architecture des systèmes P2P fondés sur DHT

Comme montré par la Figure 5.2, les systèmes P2P fondés sur DHT sont généralement structurés en 3 couches.

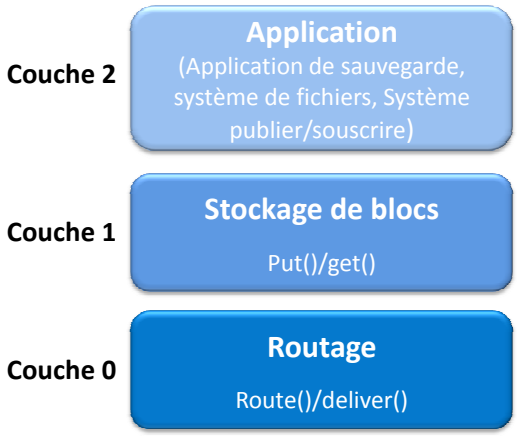


Figure 5.2 – Architecture des applications distribuées basées DHT

1. La couche de routage : elle est appelée aussi KBR (Key-Based Routing) puisqu'elle définit un protocole de routage à base de clé. Cette couche permet de cacher la complexité de l'infrastructure par des protocoles de routage et des algorithmes de tolérance aux pannes pour les couches supérieures.

2. La couche de stockage de données : c'est la DHT elle-même. Le service de base d'une DHT est le stockage de données réparti, tolérant aux pannes et persistant. Une DHT peut couvrir un très grand nombre de pairs et stocker une large quantité de données. Elle fournit l'API *get* et *put* qui simplifie beaucoup la conception des applications à large échelle.

3. La couche Application : elle représente les applications ou systèmes répartis qui utilisent une DHT. Nous trouvons les applications de sauvegarde telles que PeerStore [84], les systèmes de fichiers coopératifs CFS [85], les systèmes publier/souscrire basés DHT tels que Scribe [20], les RSD [2, 3, 4, 7, 9], etc.

D'après cette architecture, le traitement de disponibilité de données pour une telle application peut se faire généralement soit au niveau de la couche de stockage de données, soit au niveau de la couche Application. La majorité des solutions existantes dans la littérature traitent le problème de disponibilité de données au niveau de la DHT.

5.3 État de l'art : traitement de la disponibilité de données

Pour les RS centralisés, les fournisseurs de services sont ceux responsables de la disponibilité de données en offrant des serveurs dédiés et des mécanismes de tolérance aux pannes. Nous étudions alors les travaux existants qui traitent le problème de disponibilité de données dans les RSD. Puis, nous nous intéressons à ce problème dans les systèmes publier/souscrire basés DHT puisque notre plateforme MULUS est fondée sur ce type de système. Enfin, nous étudions les solutions proposées pour les réseaux P2P structurés (DHT).

5.3.1 Traitement de la disponibilité de données dans les RSD

Le problème de disponibilité de données sur les RSD est un sujet d'actualité où peu de solutions sont proposées. En effet, la majorité des RSD tels que Quasar [86], PeerSoN [2] et DECENT [41] s'intéresse à la confidentialité des données. Ils reconnaissent les problèmes de

disponibilité dans RSD, cependant, ils ne proposent pas une solution pour garantir une haute disponibilité des données dans le système. En outre, les utilisateurs des RSD ont la liberté de stocker leurs contenus sur des équipements de leurs choix. En réalité, ces équipements n'ont pas nécessairement une disponibilité élevée et ils peuvent même être défaillants. Par conséquent, assurer 24/7 disponibilité des contenus des utilisateurs stockés dans un cadre décentralisé est un défi. Dans la littérature, les solutions pour les RSD qui traitent la disponibilité, présentent deux grandes classes de techniques pour assurer la disponibilité :

- nœuds stables : cette classe suppose que les équipements des utilisateurs forment une infrastructure stable et ont une disponibilité relativement élevée. Cette hypothèse restreint le choix de l'utilisateur de son système de stockage. Par exemple, les passerelles domestiques et les décodeurs ont des capacités de stockage et ils ont un taux de disponibilité plus élevé par rapport à un téléphone portable ou un ordinateur portable.
- mise en cache et réplication : les auteurs dans [87] ont montré par expérimentation l'effet de différents paramètres sur la disponibilité de données dans les RSD. Ces paramètres incluent le temps de disponibilité de l'utilisateur, le nombre de répliques et la politique de placement des répliques. Outre la disponibilité globale, les auteurs considèrent également la disponibilité du contenu seulement à leurs amis. Ils étudient différentes stratégies de réplication. Ils identifient quatre stratégies de réplication adaptées :

  1. **réplication sans contraintes :** cette stratégie est peu utilisée. Les répliques sont placées pour maximiser la disponibilité d'un contenu chez les contacts les plus actifs d'un utilisateur ou de manière aléatoire sans contraintes. Leur étude montre que le placement des répliques chez les contacts les plus actifs avec une réplication de 40% donne une haute disponibilité des contenus.
  2. **réplication fondée sur les relations de confiance :** l'utilisateur sélectionne les répliques chez d'autres utilisateurs selon leurs relations de confiances ;
  3. **réplique sélectionnée par le système :** les répliques sont hébergées chez des utilisateurs selon des paramètres choisis par le fournisseur de services ;
  4. **réplication fondée sur les intérêts des utilisateurs :** la réplication est faite selon les intérêts des utilisateurs.

Nous détaillons dans la suite de cette section les stratégies les plus utilisées par les RSD.

5.3.1.1 Nœuds stables

PrPl [4] et Vis-á-Vis [6] suivent ce modèle et proposent de répliquer les données des utilisateurs sur un nombre d'utilisateurs qui peuvent servir comme un proxy quand le propriétaire de données est hors ligne.

PrPl suppose que le RS est déployé sur les passerelles domestiques, les décodeurs, les consoles de jeux ou sur des serveurs d'hébergement confiants gratuits ou payants qui sont en ligne la plupart du temps.
Vis-á-Vis utilise des serveurs virtuels fonctionnant sur l'infrastructure d'Amazon EC2 [1] qui ont une très haute disponibilité. Cette approche évite les coûts de la bande passante et de stockage associés à la gestion des répliques.

5.3.1.2 Technique de réplication fondée sur les relations de confiance

Uniquement Safebook [3] permet à un utilisateur de créer son groupe de répliques fondé sur ses relations de confiance ou d'amitié.

5.3.1.3 Réplique sélectionnée par le système

Les données d'un utilisateur sont répliquées et cachées chez un ensemble d'utilisateurs qui fonctionnent comme un proxy quand le propriétaire de ce contenu n'est pas en ligne. Pour SuperNova et Cachet, le placement des répliques est une décision prise par le système lui-même, c'est-à-dire par les super-pairs ou les nœuds DHT, respectivement.

5.3.1.4 Réplication fondée sur les intérêts des utilisateurs

Cette stratégie est utilisée souvent par les microblogs décentralisés. Ils utilisent les modèle publier/souscrire et reproduisent les donnés des utilisateurs producteurs uniquement pour les utilisateurs souscrits. Ils suivent l'approche de *gossip* pour propager éventuellement les mises à jour des répliques. La propagation de mises à jour à long terme peut engendrer des copies éculées chez des répliques pendant quelques instants. L'approche de *gossip* peut générer un grand volume de messages sur le réseau aux moments des mises à jour par rapport aux autres approches.
Forsyth *et al.* [88] ont proposé aussi une approche pour la réplication et la propagation de

1. http ://aws.amazon.com/ec2-sla/

mises à jour des copies pour les microblogs décentralisés. Ils proposent d'envoyer une requête sur un microblog par chemin aléatoire sur le graphe social et mettre en cache les informations sur le chemin et les données si la recherche aboutit. La mémorisation des informations sur le chemin permet aux mises à jour futures d'être appliquées par le même chemin, ce qui permet d'avoir des répliques cohérentes.

Une étude récente faite par Asthna *et al.* [89] porte sur l'optimisation du nombre de répliques et leur placement dans le contexte des microblogs en P2P. Ils proposent aussi un algorithme fondé sur le *gossip* pour placer les microblogs sur une partie du réseau pour assurer une bonne disponibilité aux utilisateurs. Ce travail tente de trouver un équilibre entre le nombre d'utilisateurs où un microblog est répliqué et le nombre d'utilisateurs qui doivent être interrogés pour trouver un microblog, tout en gardant l'utilisation de la bande passante au minimum. Selon leur analyse, un microblog doit être répliqué environ 6 à 20% du nombre d'utilisateurs dans un réseau de 10.000 et 100.000 utilisateurs de microblogging respectivement, afin d'assurer une haute disponibilité tout en exigeant moins de bande passante pour les recherches futures.

5.3.1.5 Synthèse

Le tableau ci-dessous récapitule les caractéristiques des RSD actuels.

Tableau 5.1 – Disponibilité de données dans les RSD

RSD	**Disponibilité de données**	**Stratégie de placement**
Quasar	-	-
PeerSoN	-	-
FETHR	réplication	fondée sur les intérêts des utilisateurs
Safebook	réplication	fondée sur les relations de confiance
PrPl	nœuds stables	-
Cuckoo	réplication	fondée sur les intérêts des utilisateurs
Vis-á-Vis	nœuds stables	-
SuperNova	réplication	sélectionnée par le système (super pair)
Cachet	réplication et mise en cache	sélectionnée par le système (DHT)

5.3.2 Traitement de la disponibilité de données dans les systèmes publier/souscrire basés DHT

A notre connaissance, Scribe est le seul système publier/souscrire qui propose une solution au niveau de la couche 2 (Application) pour pallier le problème de churn. Il propose d'utiliser les messages de *heartbeat* pour détecter la déconnexion des nœuds et réparer l'arbre de multicast.

Périodiquement, chaque nœud non-feuille de l'arbre de multicast envoie un message *heartbeat* à ses nœuds fils. Lorsque les évènements sont fréquemment publiés sur un tel sujet, Scribe cesse d'utiliser les messages de *heartbeat* puisque les évènements servent eux-même de message *heartbeat* implicite. Un nœud fils détecte que son parent est défectueux quand il ne parvient pas à recevoir des messages de *heartbeat*. Lors de la détection de l'échec de son parent, le nœud fait appel à la DHT (*Pastry*) pour acheminer le message de souscription (*subscribe*) à l'identifiant du nœud responsable du sujet de la souscription. Pastry achemine le message de souscription vers un nouveau parent pour la réparation de l'arbre de multicast.

Comme l'exemple montré par la Figure 5.3, le nœud 1001 détecte la déconnexion du nœud 1101. Il utilise donc Pastry pour router le message de souscription vers le root 1100 à travers d'autre(s) nœud(s). Ce message atteint le nœud 1111, qui ajoute 1001 à sa table des souscrits et, comme il n'est pas un root, il envoie un message *subscribe* vers la racine. Par conséquent, le nœud 1100 ajoute 1111 à sa table des souscrits.

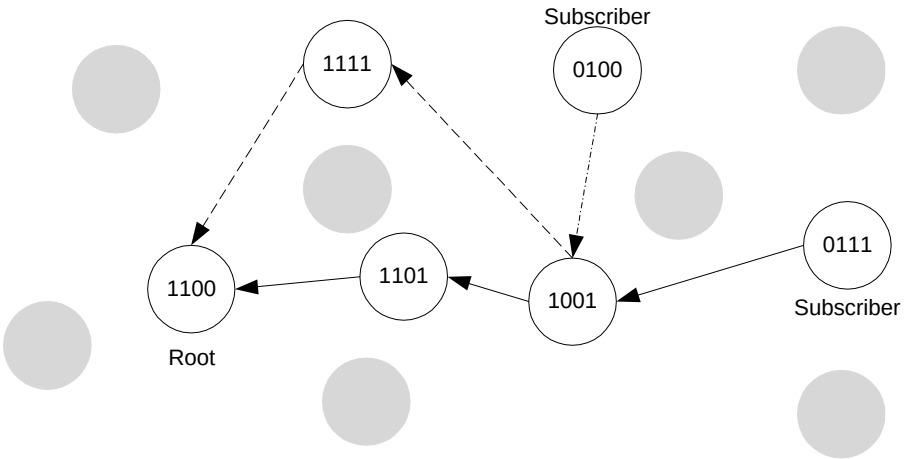


Figure 5.3 – Réparation de l'arbre de Multicast de Scribe

Par expérimentation, nous trouvons que l'utilisation des messages de *heartbeat* est insuffisante pour assurer la disponibilité des souscriptions. En effet, les notifications ne seront plus acheminées aux souscrits à partir de 30 connexions/déconnexions par seconde comme montré par la courbe de la Figure 5.4. Nous pouvons alors confirmer que cette solution est insuffisante pour assurer la disponibilité de données pour les RS à large échelle présentant un taux de churn élevé.

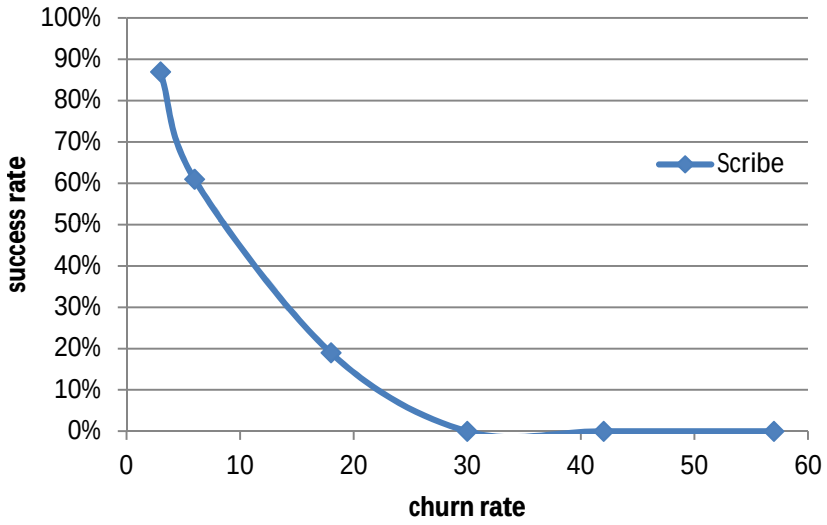


Figure 5.4 – Dégradation de la disponibilité des souscriptions en augmentant le taux de churn

5.3.3 Traitement de la disponibilité de données dans les réseaux P2P structurés

Rodrigues et Blake [90] ont montré que le stockage de grandes quantités de données sur une DHT classique n'est pertinent que si la durée de connexion des pairs est de quelques jours en moyenne. Les premières recherches effectuées pour pallier le problème de connexions/déconnexions des pairs se sont adressées à la couche de routage (montrée par la Figure 5.2) pour maintenir la cohérence du voisinage logique par la reconfiguration des pairs [83, 91].
En état de churn (fréquence de connexion/déconnexion est élevée), ces solutions ne sont plus suffisantes pour la conservation des données. L'intervention au niveau de la couche 1 (stockage de données) doit avoir lieu pour remédier à ce problème. D'autres solutions sont alors proposées au niveau de la couche de stockage de données. Le principe est le même pour toutes ces solutions. Il s'agit de proposer une méthode de réplication des blocs de données.

Il y a deux protocoles de réplication :

- le premier est le protocole de placement initial proposant deux stratégies :
  - réplication sur les voisins (leafset) ou sur les successeurs ;
  - réplication à clés multiples.
- le deuxième est le protocole de maintenance.

Nous détaillons dans la suite chacune de ces solutions.

5.3.3.1 Réplication à base de leafset ou successeurs

La réplication à base de leafset est implémentée sur Pastry [19] et DHash [92]. Selon cette stratégie, le nœud root responsable d'un bloc de données crée une copie (*replica*) et l'envoie à un ensemble de ses voisins (leafsets) : ses successeurs et ses prédécesseurs sur le cercle de la DHT (voir Figure 5.5).

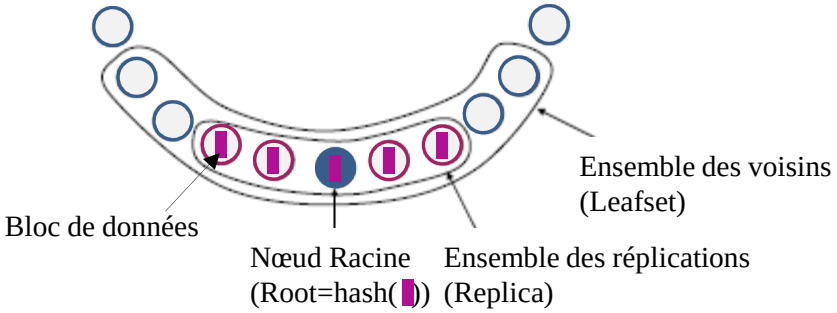


Figure 5.5 – Réplication à base de leafset (liste des voisins)

La réplication sur les successeurs suit le même principe que la réplication sur les voisins sauf que les blocs de données sont répliqués uniquement sur les successeurs immédiats du nœud responsable.

Pour ces deux variantes de stratégies, l'ensemble des réplications est localisé sur des nœuds proches et de la même zone. Ainsi, cette solution demande des contraintes fortes pour le placement des réplications afin d'éviter les zones des nœuds dynamiques. En effet, si la réplication se fait sur les nœuds voisins et ces derniers sont tous dynamiques, alors la solution reste inefficace.

Pour les systèmes publier/souscrire, cette proposition reste valide si uniquement le nœud Rendez-Vous (responsable d'une souscription) est dynamique. Si les nœuds intermédiaires entre les souscrits et le root sont aussi dynamiques, alors cette méthode de réplication simple

n'est plus suffisante car elle ne permet pas de garder tout le chemin entre le root et les souscrits.

5.3.3.2 Réplication à base de clés multiples

Cette solution propose de calculer k clés différentes à partir de la clé d'origine pour chaque bloc de données. Chaque bloc de données est recopié sur k autres nœuds racines correspondants aux k clés retrouvées. La Figure 5.6 montre le principe de cette solution.

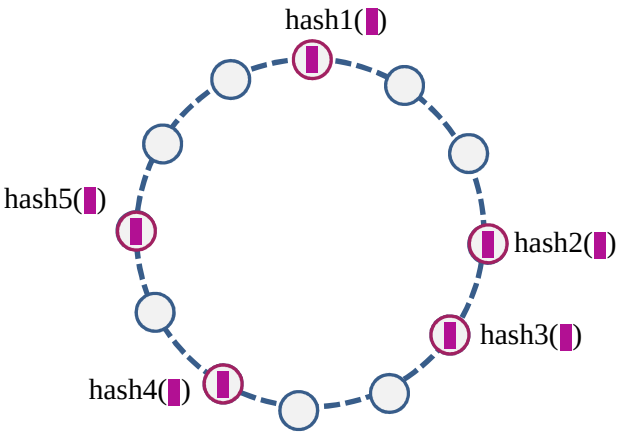


Figure 5.6 – Réplication sur la liste des voisins

Cette solution est implémentée sur CAN [47] et Tapestry [49]. Le coût de calcul des clés limite la performance du système conçu surtout avec l'augmentation du nombre de blocs de données. D'autre part, si les nœuds qui forment le chemin de la souscription sont en état de churn, le problème n'est plus résolu par la réplication des nœuds Rendez-Vous.

5.3.3.3 Protocole de maintenance

Les protocoles de maintenance s'occupent de maintenir k copies de chaque bloc de données en respectant la stratégie de placement de DHT. C'est-à-dire que les k copies de chaque bloc de données doivent dans tous les cas être enregistrées sur les pairs voisins de la racine du bloc dans le cas d'une stratégie de réplication à base de leafset, et sur les pairs racines pour le schéma de réplication par clés multiples.

Le mécanisme de maintenance à base de leafset se base sur l'échange périodique d'information entre les voisins. Par exemple, avec l'algorithme de maintenance de PAST [93],

chaque pair envoie un *bloom filter* des blocs qu'il maintient dans son leafset. Lorsqu'une telle requête est reçue par un pair des voisins, ce dernier utilise son *bloom filter* pour déterminer s'il stocke un ou plusieurs blocs que l'envoyeur doit également stocker. Il répond ensuite avec la liste des clés de ses blocs. Le pair à l'origine de la requête peut par la suite récupérer les blocs manquants trouvés dans l'ensemble des réponses à ses requêtes.

Pour la stratégie de réplication à base de clés multiples, la maintenance doit se faire séparément pour chaque bloc de données. Pour chacun stocké dans le système, la vérification de ses pairs racines est nécessaire périodiquement pour s'assurer qu'ils sont toujours opérationnels et maintiennent encore une copie du bloc de données.

5.4 Traitement de la disponibilité de données dans MULUS

Nous rappelons que l'architecture de notre plateforme utilise des dispositifs sélectionnés de chaque cluster pour jouer le rôle de serveurs d'évènements. Cette architecture met les producteurs et les consommateurs en collaboration sans recours à une couche de brokers étrangers (voir Figure 2.14).

La disponibilité de données partagées est un critère de performance très important pour un tel système de communication. Ainsi, la connexion et déconnexion rapides et successives avec des durées de sessions très variées perturbent l'acheminement des messages. Bien que les DHT assurent l'auto-organisation des nœuds, la disponibilité des données reste un défi pour les RSD, en particulier celui que nous avons proposé fondé sur un système publier/souscrire P2P. En effet, le dynamisme fort pour des nœuds responsables d'un très grand nombre de souscriptions perturbe le routage des publications. En fait, la perte de données sur un système publier/souscrire est due essentiellement à la perte des souscriptions dont le rôle consiste à mémoriser les besoins des utilisateurs dans le service d'évènements distribué. Pour cela, nous proposons de répliquer les souscriptions qui ne représentent qu'une description des besoins des utilisateurs et non les données partagées. Cette réplication ne fait pas perdre de mémoire des ressources des utilisateurs et permet d'acheminer les publications sans perte.

Notre mécanisme défini pour assurer la disponibilité de données se base sur la réplication des souscriptions pour créer différents chemins pour l'acheminement des publications. Dans la

suite, nous proposons deux stratégies de réplication des souscriptions. La première réplique la souscription à partir d'un autre point d'accès le plus éloigné du point d'accès courant. Ceci permet de créer un chemin pour la souscription complètement différent du premier, ce qui permet de retrouver un chemin pour l'acheminement des publications. Pour trouver le chemin de réplication, nous définissons une nouvelle approche appelée complémentaire des digits de poids forts appliquée sur les identifiants des nœuds DHT. La deuxième stratégie fait la réplication des souscriptions sur d'autres nœuds selon leur disponibilité en cherchant à atteindre une disponibilité totale souhaitée par l'utilisateur. Ces deux stratégies de réplication sont détaillées dans les sous-sections suivantes.

5.4.1 Complémentaires des digits de poids forts

La première solution proposée consiste à dupliquer les souscriptions sur des nœuds géographiquement distants en se basant sur le complémentaire des digits identifiant de la souscription. Par ailleurs, sur DHT, les nœuds sont généralement arrangés sur un cercle allant de 0 à 2^{128} selon les identifiants attribués aux différents nœuds. Ainsi, le complémentaire du digit du poids fort d'un identifiant de nœud permet de se placer sur l'arc opposé du cercle pour s'éloigner au maximum de la zone d'origine pouvant présenter un dynamisme fort (voir Figure 5.7).

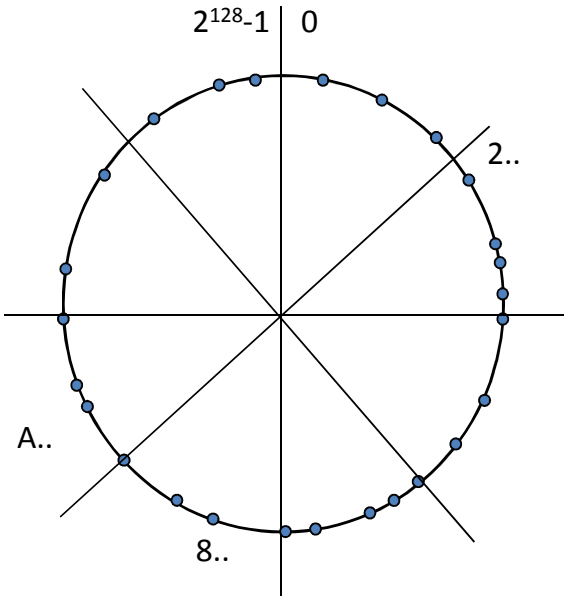


Figure 5.7 – Identification des nœuds sur le cercle DHT pour la réplication

Pour trouver le complémentaire du digit, nous proposons la formule : $\overline{D_i} = (D_i+8)mod16$, avec D_i étant le digit d'occurrence i de l'identifiant du nœud [94]. Nous commençons par le digit de poids fort de l'identifiant d'un nœud pour trouver l'identifiant d'un nœud placé sur l'arc opposé. Comme dans l'exemple montré par la même figure, un nœud dont l'identifiant commence par A a comme nœud opposé sur le cercle celui dont l'identifiant commence par 2.

Nous proposons alors l'Algorithme 3 pour la sélection de plusieurs nœuds pour les répliques. Le nombre de répliques peut être choisi soit par les utilisateurs selon l'importance de leurs besoins sous le contrôle de l'administrateur du système. À chaque réplication, nous choisissons un nouveau point d'accès pour envoyer de nouveau la souscription selon le complémentaire du digit suivant, du plus fort au plus faible.

Algorithme 3 : réplication des souscriptions fondée sur le complémentaire du digit pour la sélection des nœuds

```
   Data : byte nodeID[128], integer Nb_Rep, Subscription S
   Result : byte nodeID_Rep[128]
 1 begin
 2     i, c : integer;
 3     for c ← 1 to Nb_Rep do
 4         for i ← 1 to c-1 do
 5             ;            // construire la première partie de l'ID du nœud de
                réplication
 6             nodeID_Rep[i] ← nodeID[i];
 7         end
 8         nodeID_Rep[c] ← (nodeID[c] + 8)modulo16;   // complémentaire du digit
 9         for i ← c + 1 to 128 do
10             ;            // construire la deuxième partie de l'ID du nœud de
                réplication
11             nodeID_Rep[i] ← Math.random() ∗ 16;
12         end
13         subscribe(S,nodeID_Rep) ; ;                  // répliquer la souscription
14     end
15 end
```

En appliquant cet algorithme, le premier point d'accès est responsable de dupliquer la souscription en se basant sur le digit complémentaire (Ligne 8, Algorithme 3) pour retrouver les nœuds de duplication les plus loin. La duplication multiple est faite en utilisant les différents digits composant l'identifiant du nœud en commençant par le poids le plus fort.

Par cette méthode de duplication, nous nous assurons que la publication peut retrouver un chemin complet pour arriver aux utilisateurs intéressés même au moment de la déconnexion des serveurs intermédiaires.

La Figure 5.8 illustre l'application de cet algorithme pendant la phase de souscription. Comme montré par cette même figure, le premier point d'accès d'identifiant *ABD5...* achemine la souscription vers le root *B5DC...* en créant un chemin entre ces deux nœuds qui sera par la suite utilisé pour l'acheminement des publications. Ensuite, nous cherchons le nœud le plus loin (opposé) par rapport au premier point d'accès en utilisant le complémentaire du digit de poids fort (A). Enfin, il réplique cette même souscription à travers le nœud 2* trouvé par le complémentaire (Ligne 13, Algorithme 3). Enfin, si le nombre de réplications est non atteint, nous cherchons un autre nœud par le complémentaire du digit suivant et nous faisons la même chose jusqu'à atteindre le nombre souhaité de réplications. Ceci nous permet de créer différents chemins pour la souscription pour atteindre le root comme montré par cette même figure. Ces chemins sont assez loin l'un de l'autre ce qui permet aux publications d'arriver aux souscrits même si un ou plusieurs nœuds ne sont pas disponibles.

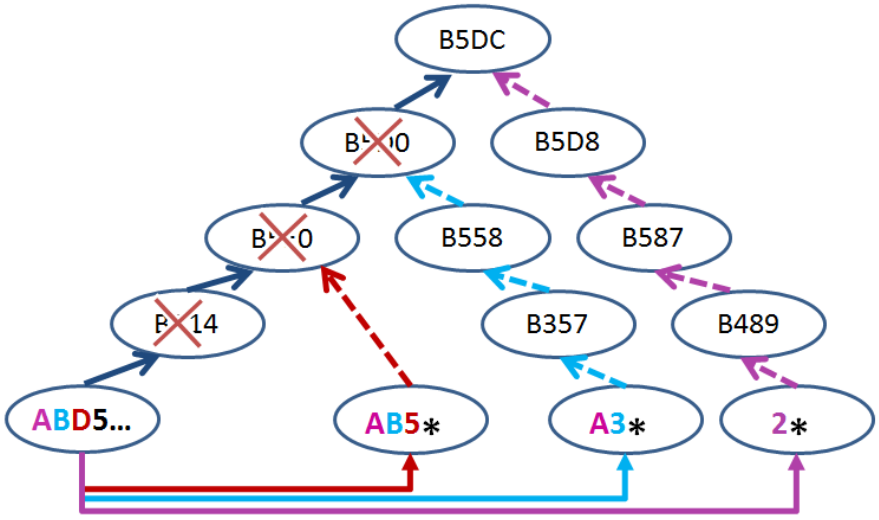


Figure 5.8 – Réplications avec le complémentaire des digits

5.4.2 Réplication selon la disponibilité des nœuds

Notre deuxième stratégie est fondée sur une réplication selon la disponibilité des nœuds. En effet, la disponibilité de données est liée essentiellement à la disponibilité des nœuds. En

effet, un nœud est dit disponible s'il peut remplir la mission ou la fonction pour laquelle il a été conçu à un instant donné ou pendant un intervalle de temps donné.
Le principe de cette approche consiste alors à vérifier le taux de disponibilité du chemin créé pendant la phase de souscription pour réduire le nombre de chemins répliqués et avoir en même temps le chemin le plus fiable. En effet, ce chemin présente le chemin d'acheminement des notifications pendant la phase de publication. Pour ce faire, nous supposons que l'identifiant d'un nœud sur DHT reflète le taux de leur disponibilité. Chaque nœud DHT possède une valeur de disponibilité notée a.
Cette valeur de disponibilité a représente la durée de la disponibilité du nœud pour la réception d'un flux de messages. Cette durée dépend du taux de churn du parent dans l'arbre de multicast. Cependant, dans le cas où un parent se déconnecte, une telle DHT possède ces propres mécanismes de réparation pour rétablir la situation du fils, c'est-à-dire le fils qui perd un parent, s'attache automatiquement à un nouveau parent. Mais, à chaque déconnexion d'un nœud parent, il faut un temps, noté Δ, pour s'attacher à un autre parent et continuer à recevoir le flux de messages. Alors pour une telle déconnexion, la durée de la non-disponibilité du nœud pour la réception du flux correspond exactement au temps Δ (*Indisponibilité = 1 - Disponibilité*). Ainsi, la disponibilité de réception de flux a peut être décrite par la formule 5.1.

$$a = \frac{\text{unité de temps de churn} - \text{nombre de churn du parent du noeud} * \Delta}{\text{unité de temps de churn}} \tag{5.1}$$

Par exemple, si le churn du parent d'un nœud est de 1 déconnexion par 1 heure, et pour s'attacher à un autre parent demande 30 secondes, alors la disponibilité a de ce nœud vaut :

$$a = \frac{3600 - 1 * 30}{3600} = 0.991 \tag{5.2}$$

Ainsi, pour chaque nœud qui s'ajoute au chemin de la souscription, la valeur de disponibilité a du nœud est calculée. Par conséquent, chaque branche de l'arbre de multicast (chemin du souscrit au root définit par l'ensemble des nœuds parcourus), a aussi une valeur de disponibilité bien déterminée. Pour calculer cette valeur, nous appliquons la règle de disponibilité pour un système en série. Il suffit alors de multiplier les valeurs de disponibilité de tous les nœuds formant la branche en question pour trouver la disponibilité totale donnée par la formule 5.3. Ceci nous permet d'associer une valeur de disponibilité à chaque chemin utilisé

pour acheminer une souscription de l'utilisateur vers le nœud root responsable de mémoriser cette souscription.

$$A = \prod_{i=1}^{n}(a_i) \tag{5.3}$$

Avec :

- A : disponibilité totale du chemin ;
- a_i : disponibilité d'un nœud calculé par la formule 5.1 ;
- n : le nombre de nœuds parcourus par la souscription.

Donc, si par exemple un fils possède 5 parents pour s'attacher à la racine avec une même valeur de disponibilité pour tous les parents qui vaut 0.991 donnée par la formule 5.2. Alors, la disponibilité totale A est le produit de disponibilité de tous ces nœuds parents comme exprimé par la formule suivante :

$$A = \prod_{i=1}^{5}(0.991) = 0.955 \tag{5.4}$$

Afin de garantir la disponibilité des données, nous fixons un taux de disponibilité accepté, dit aussi taux de disponibilité souhaité, à atteindre pour assurer la fiabilité totale de notre plateforme. Ce taux est noté A_{souh}. Par conséquent, nous cherchons à faire des réplications de souscriptions si le taux souhaité n'est pas encore atteint. Pour ceci, nous comparons la valeur de disponibilité atteinte A par rapport à A_{souh}. Si la valeur de disponibilité du chemin est supérieure ou égale à la valeur de disponibilité souhaitée, alors ce chemin est maintenue et considéré comme un chemin fiable assurant la disponibilité souhaitée des données. Dans le cas où la valeur de disponibilité de chemin est strictement inférieure à la valeur de disponibilité souhaitée, le chemin est considéré comme insuffisant et un deuxième chemin doit être établit pour atteindre ou dépasser A_{souh}. Par la suite, nous cherchons un chemin complémentaire ayant une disponibilité que nous appelons disponibilité complémentaire pour atteindre la valeur de disponibilité totale souhaitée. En utilisant la formule de calcul de disponibilité pour un système en parallèle, nous devons vérifier l'égalité donnée par l'équation 5.5. En effet, les chemins de réplication sont considérés comme des composants liés en parallèle puisque l'un de ces chemins permet d'acheminer une publication du root au souscrit.

$$1 - (1 - A)(1 - A_{comp}) = A_{souh} \tag{5.5}$$

Avec :

- A : taux de disponibilité du chemin parcouru calculé par la formule 5.3 ;
- A_{comp} : taux de disponibilité du chemin complémentaire ;
- A_{souh} : taux souhaité de la disponibilité total.

Ensuite, le chemin complémentaire est choisi par son premier nœud (nœud fils) ayant la disponibilité supérieure ou égale ou la plus proche de A_{comp}. D'après 5.5, A_{comp} vaut :

$$A_{comp} = \frac{A - A_{souh}}{A - 1} \tag{5.6}$$

Enfin, nous cherchons l'identifiant du nœud complémentaire, correspondant à la valeur de la disponibilité complémentaire trouvée par 5.6. Pour ce faire, nous convertissons A_{comp} à un identifiant exprimé dans la base hexadécimale en utilisant la formule 5.7. Une fois retrouvé, nous répliquons la souscription à partir du nœud d'identifiant ID_{DHT_comp} et nous calculons le taux de disponibilité totale de tous les chemins créés. Si le taux total, noté A_{tot}, calculé par la formule 5.8 (disponibilité d'un système en parallèle) est encore inférieur à la disponibilité souhaitée, nous répétons la même démarche jusqu'à atteindre le taux souhaité.

$$ID_{DHT\_comp} = [A_{comp} * 2^{128}]_{Hexa} \tag{5.7}$$

$$A_{tot} = 1 - \prod_{i=1}^{n}(1 - A_i) \tag{5.8}$$

Avec :

- A_i : la disponibilité du chemin parcouru calculée par 5.3 ;
- n : le nombre de chemins créés ;
- A_{tot} : la disponibilité totale de tous les chemins créés.

5.5 Évaluation des performances

Ces approches sont implémentées sur le système publier/souscrire Scribe intégrant le simulateur Freepastry et fondé sur la DHT Pastry. Nous faisons des expérimentations pour la validation de la première stratégie de réplication dans une première partie puis une comparaison entre les deux stratégies proposées dans une deuxième partie.

5.5.1 Évaluation de la première approche

L'expérimentation est faite avec un réseau de 1000 nœuds dynamiques en tenant compte des métriques suivantes :

- **le taux du dynamisme fort** : il présente le nombre de connexion/déconnexion par minute ;
- **le taux de succès de l'acheminement des publications :** c'est le rapport entre le nombre de publications correctement acheminées et le nombre des souscriptions, de cette même publication, en attente. Autrement dit, c'est le rapport entre le nombre des souscrits notifiés et le nombre total des souscrits. Cette métrique permet ainsi de nous indiquer le taux de succès de la réception des notifications ;
- **le trafic :** il présente le nombre de messages échangés pour la réplication des souscriptions.

La période de simulation est fixée à 340 secondes avec des temps de session de 30 secondes. La moyenne de connexion/déconnexion est de 3 à 57 login/logout par minute. Nous avons touché tous les nœuds du réseau pour assurer l'effet du dynamisme. Toutes les 60 secondes, il y a de nouvelles publications et souscriptions sur le réseau. Chaque simulation est répétée 10 fois pour prendre la moyenne des différentes métriques trouvées.

Nous observons le taux de succès par rapport au taux du dynamisme avec variation des nombres de réplications. La courbe de la Figure 5.9 montre le taux moyen de réussites observées durant l'application de l'approche pour différents taux de dynamisme. Nous remarquons qu'à partir d'un taux de 30 connexions/déconnexions par minute, Scribe (avec 0 référence) ne peut plus résister et les tables de routage des souscriptions ne sont plus disponibles. En effet, la résistance à un taux moins de 30 connexions/déconnexions par minute est assurée par l'auto-organisation de Scribe.

Ces courbes montrent aussi que plus le taux de churn augmente, plus nous avons besoin d'ajouter des réplications pour améliorer le taux de disponibilité et par la suite le taux de succès.

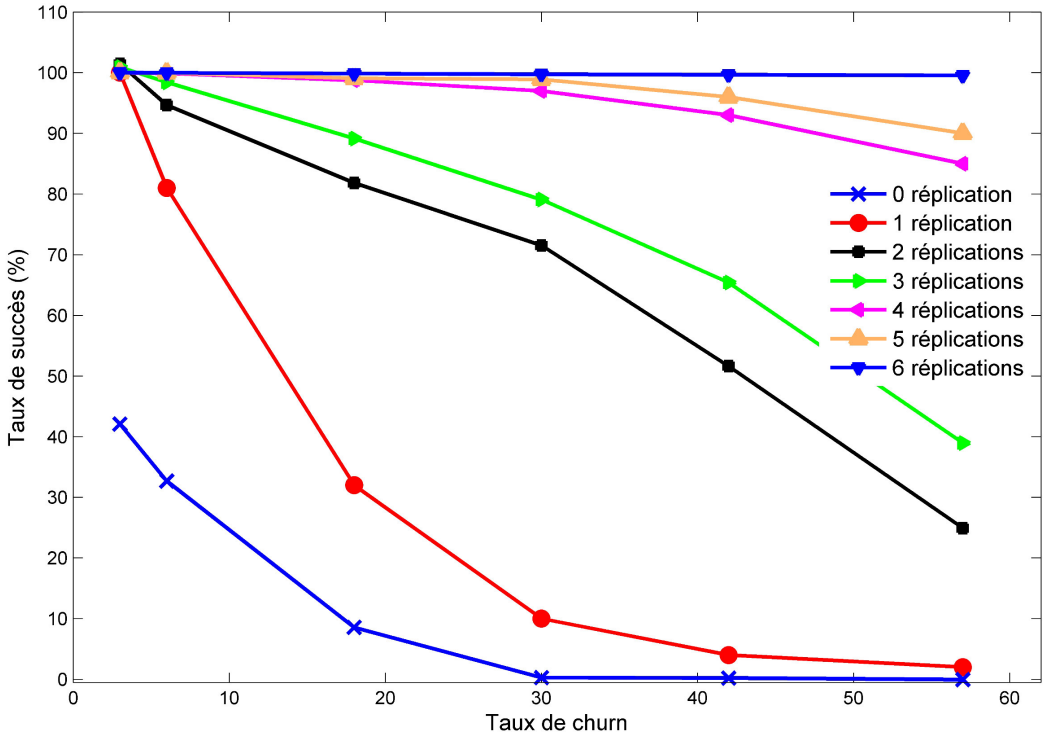


Figure 5.9 – Comparaison du taux de réussite en fonction du taux de churn

5.5.2 Comparaison entre les deux stratégies de réplication

Avec les mêmes métriques définies dans la section précédente, nous cherchons à comparer la performance des différentes approches en cherchant un taux de perte nul (0%) : réplication aléatoire, réplication fondée sur le complémentaire du digit et réplication selon le taux de disponibilité souhaité. La courbe de la Figure 5.10 montre que la réplication avec un choix aléatoire de nœuds augmente considérablement le trafic sur le réseau pour atteindre ce taux. En effet, nous devons faire plus de réplications avec cette stratégie pour atteindre ce taux et le nombre de nouvelles branches de l'arbre de multicast augmente en conséquence.

Par comparaison avec cette courbe, nous retrouvons que le trafic résultant du référencement par le complémentaire des digits est beaucoup moins faible. Ceci est expliqué par le fait que les premières réplications nous mènent à des branches bien écartées de la première ce qui améliore le taux de succès d'acheminement.

Quand à la troisième stratégie (par calcul de disponibilité souhaitée), nous retrouvons un trafic très faible par rapport aux deux premières stratégies. Ceci étant vrai puisqu'avec cette stratégie, nous fixons un taux souhaité égal à 1 et nous cherchons à atteindre ce taux

au moment de l'acheminement des souscriptions. Par conséquent, nous faisons moins de réplications pour atteindre ce taux au moment de l'acheminement des notifications.
De plus, la pente de ces trois courbes montre que le trafic augmente légèrement avec le churn pour nos deux stratégies alors qu'il est plus croissant pour la stratégie de réplication aléatoire.

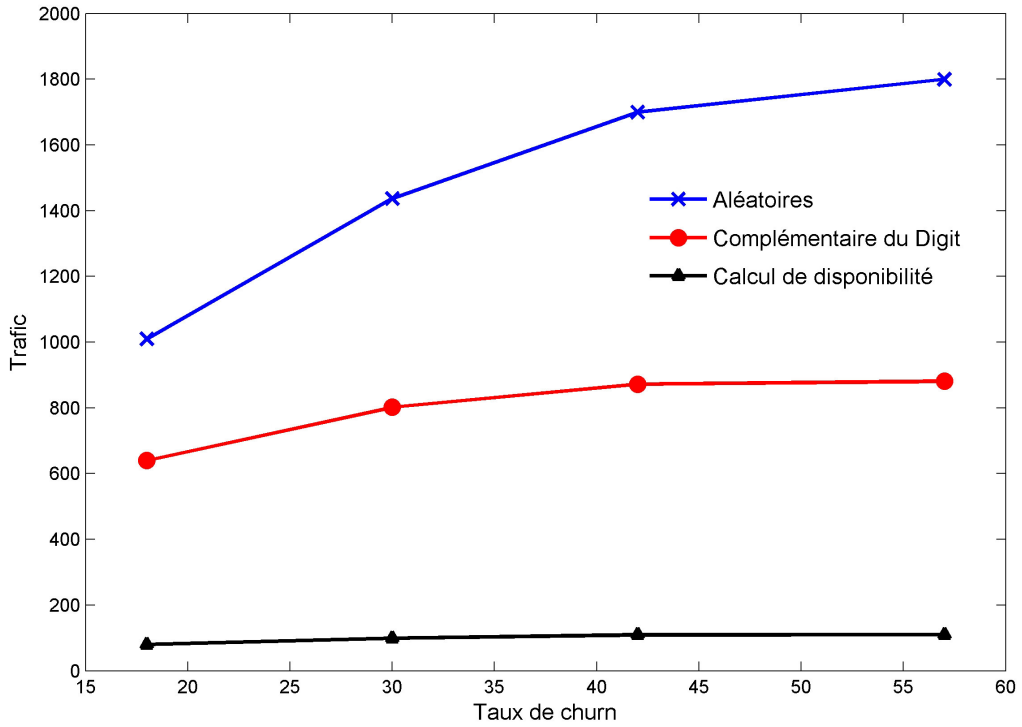


Figure 5.10 – Comparaison du trafic pour les différentes stratégies pour un taux de perte de 0% : réplication aléatoire, réplication fondée sur le complémentaire du digit et réplication selon le taux de disponibilité souhaité

5.5.2.1 Métriques de test

Afin de mieux valider la deuxième stratégie proposée et mettre en œuvre son optimisation dans la recherche des chemins de réplication en termes de temps et nombre de messages échangés, nous étudions les métriques suivantes qui seront utilisées dans notre comparaison entre les deux stratégies de réplications proposées :

– **coût de stabilisation** : le coût de stabilisation est relativement lié au nombre de souscripteurs ajoutés pour atteindre un taux espéré. En effet, et comme décrit dans la section 5.4.2, pour s'assurer de la fiabilité du chemin de souscription, nous fixons la valeur de disponibilité souhaitée. Alors, une fois la valeur de disponibilité calculée

pour un chemin sélectionné est inférieure à celle souhaitée, un nouveau chemin doit être recherché et ajouté jusqu'à atteindre la disponibilité souhaitée.

- **temps de stabilisation** : le temps de stabilisation reflète le temps mis pour que le taux de disponibilité atteigne le taux de disponibilité souhaité. Cependant, et comme mentionné pour le coût de stabilisation, on utilisera ce critère, pour déduire le temps nécessaire pour un souscripteur, pour trouver son nœud complémentaire.
- **le taux d'erreur de stabilisation** : cette métrique est exprimée par la différence entre le taux de disponibilité souhaité et le taux atteint après stabilisation. En effet, pour une valeur de disponibilité souhaitée fixée, le taux d'erreur de stabilisation, sera la marge entre la valeur complémentaire (obtenue par la formule 5.6) et celle trouvée réellement en franchissant le chemin sélectionné par le calcul.

5.5.2.2 Comparaison entre les deux stratégies de réplication

Pour évaluer notre stratégie de réplication, nous essayons de comparer la stratégie fondée sur la disponibilité avec la stratégie de référencement qui se base sur le digit complémentaire. En fait, la première consiste à dupliquer le chemin de souscription en choisissant un chemin complètement différent sans tenir compte de la valeur de disponibilité des nœuds impliqués. Alors que la deuxième stratégie fait la réplication des souscriptions en se basant sur le taux de disponibilité des nœuds. Pour comparer ces deux approches, nous fixons un taux de disponibilité souhaité à chaque fois et nous cherchons le nombre de réplications effectuées pour atteindre ce taux avec un taux de churn de 60 connexion/déconnexion par minute. Nous calculons la disponibilité totale pour les deux approches en utilisant la formule 5.8 et si la disponibilité souhaitée n'est pas atteinte par l'une des deux stratégies, alors nous faisons la réplication jusqu'à atteindre la valeur souhaitée. Nous trouvons les deux courbes de la Figure 5.11.

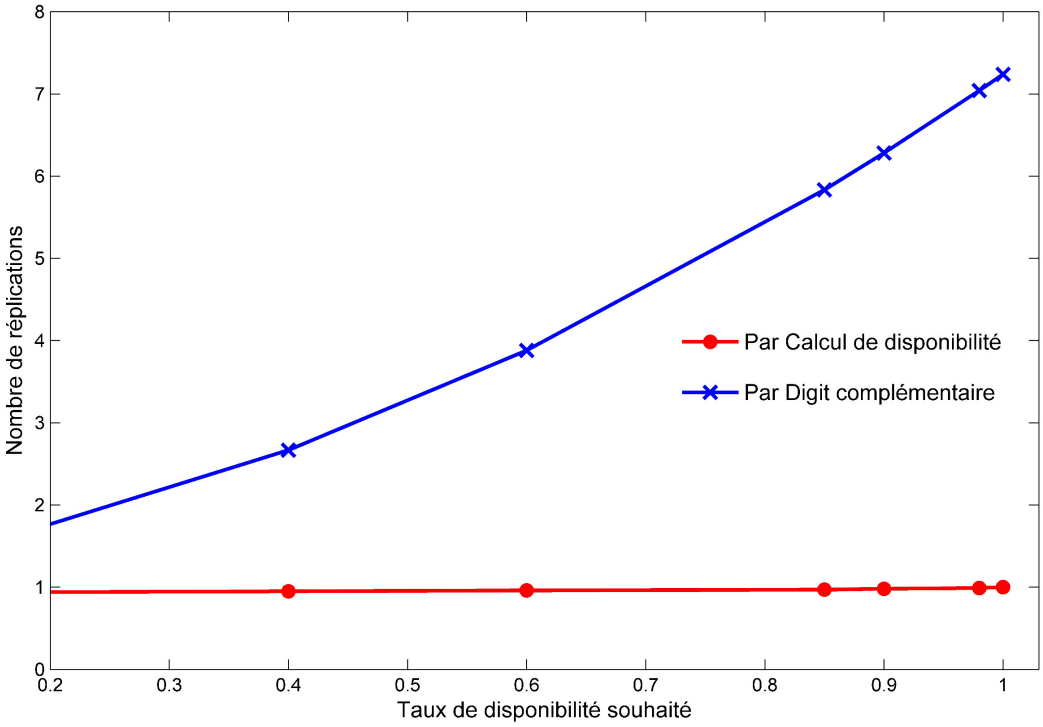


Figure 5.11 – Nombre de réplications nécessaires pour atteindre la disponibilité totale souhaitée par les deux approches proposées

Pour un taux de disponibilité souhaité faible (0.2), il nous faut 2 réplications pour atteindre ce taux par la réplication par digit complémentaire (stratégie 1). Alors qu'une seule réplication par le calcul de disponibilité complémentaire (stratégie 2) est suffisante pour atteindre ce même taux. Ainsi, nous obtenons par des expérimentations répétées une courbe presque stable avec la stratégie de réplication selon la disponibilité (stratégie 2). En effet, à la recherche de la valeur de disponibilité complémentaire, une seule réplication est souvent suffisante pour atteindre la valeur de disponibilité souhaitée. Alors qu'un nombre de réplications croissant pour la stratégie de réplication par digit complémentaire. En fait, la valeur de disponibilité obtenue par référencement ne permet pas d'atteindre le taux de disponibilité souhaité atteint par la deuxième approche. Du coup, il faut ajouter d'autres réplications pour atteindre la disponibilité totale souhaitée.

La deuxième courbe d'évaluation est illustrée par la Figure 5.12. Elle représente la variation de temps mis en fonction de la valeur de disponibilité souhaitée. Cependant, la réplication par calcul de disponibilité prend presque le même temps pour atteindre le taux espéré. Ceci s'explique par les calculs faits pour retrouver la disponibilité souhaitée et les

identifiants des nœuds pour la réplication. Donc, quelle que soit la valeur de disponibilité souhaitée, le temps de stabilisation est presque constant, car une seule référence est souvent suffisante pour atteindre la valeur désirée.

Comparant avec la première stratégie, le temps de stabilisation varie selon la valeur de disponibilité espérée. Il dépend de la valeur de disponibilité des nœuds sur lesquels nous mettons les répliques. Chaque fois que le taux de disponibilité augmente, le nombre de réplications effectuées augmente, et par la suite le temps de stabilisation augmente.

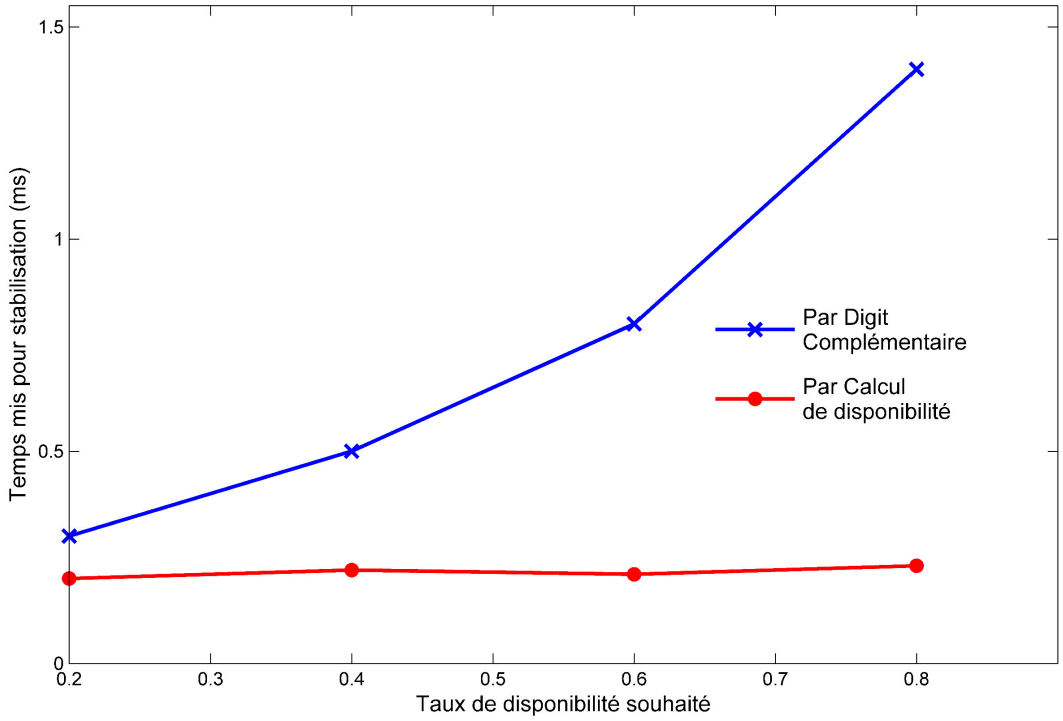


Figure 5.12 – Temps mis pour la stabilisation en fonction de la disponibilité souhaitée

Nous finalisons notre évaluation par le taux d'erreur qui varie selon le nombre de nœuds. Nous constatons d'après la courbe de la Figure 5.13 que le taux d'erreur avec réplication selon la disponibilité est inférieur à celui d'une réplication par digit complémentaire. Ceci est expliqué par le fait que la première stratégie se base sur le choix de l'identifiant du nœud reflétant la disponibilité complémentaire. Par contre, dans l'approche de référencement fondé sur le digit complémentaire, le taux d'erreur est plus important car la disponibilité atteinte réellement est généralement inférieure à celle souhaitée.

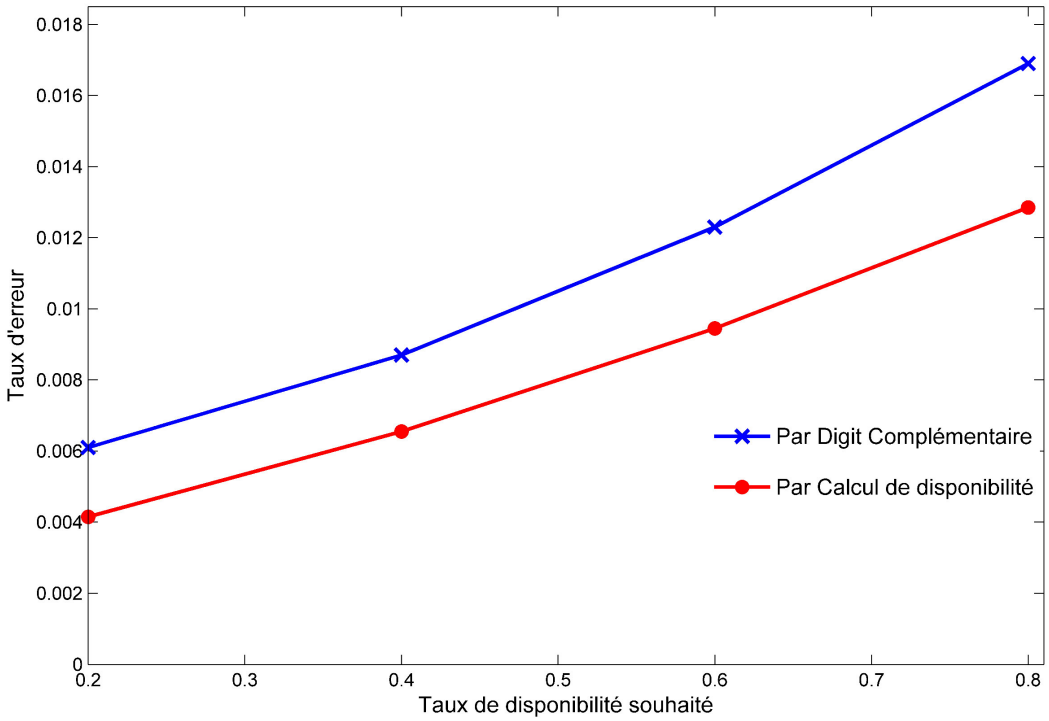


Figure 5.13 – Taux d'erreur de stabilisation en fonction de la disponibilité souhaitée

5.6 Conclusion

Le problème de disponibilité de données est survenu récemment dans les RS et en particulier ceux de topologie P2P structurée (DHT) auxquels nous nous intéressons dans cette thèse. D'après notre étude bibliographique, il y a peu de solutions définies pour pallier ce problème sur les RS de topologie P2P structurée. Il s'agit généralement de conserver des copies des blocs de données sur des nœuds distincts selon différents schémas de réplication proposés. Malheureusement, ces solutions restent insuffisantes pour les systèmes publier/souscrire basés DHT que nous avons proposés d'intégrer. En effet, l'effet de churn touche les nœuds roots et les nœuds qui forment le chemin de la souscription, chemins vitaux pour acheminer les publications aux consommateurs.

Nous avons alors proposé deux stratégies de réplication des souscriptions pour créer différents chemins pour l'acheminement des publications. La première stratégie réplique la souscription à partir d'un autre point d'accès (le plus loin du point d'accès courant). Ceci permet de créer un chemin pour la souscription complètement différent du premier ce qui

permet de retrouver un chemin pour l'acheminement des publications. Pour retrouver le chemin de réplication, nous avons défini une nouvelle approche dite complémentaire des digits de poids forts appliquée sur les identifiants des nœuds DHT. La deuxième stratégie fait la réplication des souscriptions sur d'autres nœuds selon leurs disponibilités en cherchant à atteindre une disponibilité totale souhaitée par l'utilisateur. Enfin, nous avons évalué ces deux approches par expérimentation et nous avons prouvé leur efficacité.

Conclusion générale

Synthèse

Dans cette thèse, nous avons développé MULUS, un RSD pour une consommation sélective de contenus composites. L'architecture proposée de MULUS utilise un système publier/souscrire d'architecture P2P structurée. Chaque serveur d'évènements de ce système intègre une *Home Box Entity* (HBE) et est déployé sur un dispositif de l'utilisateur. Ces entités collaborent entre elles pour répondre aux intérêts des utilisateurs et pour assurer une bonne qualité de service d'un RSD.

Les principales contributions de cette thèse sont :

- le traitement de la sémantique des requêtes sur un RSD d'architecture P2P structurée : le but est de répondre aux besoins exacts des utilisateurs et d'éviter les envois de données inutiles sur les réseaux. Pour ce faire, nous avons proposé d'utiliser un système publier/souscrire d'architecture P2P structurée permettant de traiter les intérêts des utilisateurs. Pour le rendre plus sémantique, nous avons utilisé une ontologie de domaine structurée et partagée entre tous les utilisateurs. Nous avons proposé, ensuite, une méthode d'indexation de cette ontologie fondée sur les nombres premiers. Cette méthode offre un routage sémantique sur DHT. La HBE est responsable de cette ontologie pour l'utiliser pendant les phases de souscription et de publication. Elle permet aussi la mise à jour de cette ontologie pour couvrir l'apparition de nouveaux concepts liés à un domaine bien défini.
- le traitement des intérêts composites sur un RSD d'architecture P2P structurée : cette contribution est complémentaire de la première. Toujours pour mieux répondre aux besoins exacts des utilisateurs, nous cherchons à éviter les échanges qui se produisent à des moments inadéquats ou qui ne répondent pas à des intérêts composites. Nous avons

donc proposé de traiter la composition d'évènements au niveau du système publier/-souscrire utilisé. Nous avons défini une nouvelle méthode d'indexation des événements fondée sur un cube. La structure du cube assure le filtrage des évènements composites par une simple recherche binaire effectuée sur ce cube. Notre plateforme permet donc de décrire les intérêts composites des utilisateurs par des requêtes sémantiques (par ontologie) et persistantes (publier/souscrire) sur le réseau. Ceci permet de répondre aux intérêts composites des utilisateurs en temps réel et de réduire le trafic en même temps.

– le traitement de la disponibilité de données pour une livraison sans perte : nous avons proposé deux stratégies de réplication pour assurer la disponibilité des données et une livraison sans perte sur notre RSD. La première stratégie consiste à répliquer les souscriptions à partir d'un autre point d'accès, le plus éloigné du point d'accès courant, sur l'overlay DHT. Ceci permet de créer un chemin pour chaque souscription complètement différent du premier, ce qui laisse retrouver un chemin pour l'acheminement des publications. Avec cette proposition, nous devons faire assez de réplications pour atteindre un taux de perte nul. La deuxième stratégie fait la réplication des souscriptions sur d'autres nœuds selon leurs disponibilités, en cherchant à atteindre une disponibilité totale souhaitée par l'utilisateur final.

Nous avons montré l'efficacité de notre plateforme par l'implémentation des différentes propositions, en étendant le système publier/souscrire Scribe. Les résultats expérimentaux ont montré que notre approche réduit considérablement le trafic par rapport aux RS existants. D'autres métriques ont montré la scalabilité de notre plateforme vu le nombre d'utilisateurs et d'évènements supportés et le temps de routage mis pour l'acheminement des évènements. Nous avons aussi montré que la sémantique et la composition d'évènements a réduit considérablement le trafic en éliminant les échanges qui ne répondent pas aux intérêts des utilisateurs.

Concernant la disponibilité des données, nous avons comparé les deux stratégies de réplication proposées. Nous avons alors trouvé que la stratégie de réplication selon la disponibilité des nœuds est plus efficace pour minimiser le nombre de réplications, augmenter le taux de disponibilité et améliorer la qualité de service au moment de la livraison. Cependant, cette stratégie demande un temps considérable pour le calcul des taux de disponibilité de l'ensemble des chemins parcourus, surtout si nous prenons en compte la mise à jour des valeurs

de la disponibilité.

Perspectives

À court terme, nous planifions de mettre en œuvre notre plateforme MULUS pour mesurer le niveau de satisfaction des utilisateurs vis-à-vis des services offerts par notre RS. Ceci permettrait de saisir le niveau de satisfaction des consommateurs en faisant référence à un grand nombre d'utilisateurs. Nous pourrions vérifier si le RSD proposé répondrait mieux aux besoins et si les utilisateurs apprécieraient la différence par rapport aux autres RS.

À moyen terme, il serait important d'automatiser la phase de publication en intégrant un module pour l'analyse et l'annotation des contenus à publier sur MULUS. Ce module analyserait les contenus à publier et retournerait le sujet ainsi qu'une description sémantique (individu de l'ontologie de domaine) selon l'ontologie de domaine utilisée [95]. Ces informations (sujet+individu de l'ontologie) seraient utilisées pour le routage du contenu publié sur DHT afin de retrouver le nœud responsable du matching.

À long terme, des évolutions à apporter à notre plateforme seraient aussi envisageables pour pouvoir l'utiliser dans un contexte différent de celui du partage de données composites sur les RS. Autrement dit, la génération d'une description sémantique plus développée pourrait rendre cette plateforme utile dans plusieurs autres contextes. A titre d'exemples, nous citons les architectures orientées services (SOA), Universal Description Discovery & Integration (UDDI), Java Naming and Directory Interface (JNDI) et le domaine de gestion de réseaux en particulier. En effet, l'entité logicielle CECube que nous avons développée pourrait être utilisée pour la gestion des réseaux qui a besoin d'un système de notification des évènements réseau. En particulier, le SDN (Software Defined Network) a besoin d'un système de gestion scalable, puisqu'il doit avoir une vision globale de tous les équipements du réseau. Ainsi, la scalabilité pour le SDN est parmi ses grands défis. Nous pourrions alors améliorer la scalabilité de ce système de gestion en lui fournissant un système de notification distribué, qui supporte un grand nombre d'utilisateurs et filtre d'une façon distribuée et scalable les évènements reçus. Nous pourrions supposer également que les équipements réseau seraient eux-mêmes des souscrits (par exemple, un routeur n'activerait une politique de routage que dans le cas où il recevrait un évènement simple ou composite).

Liste des Publications

Revue Internationale

– A. Chaabane, C. Diop, W. Louati, M. Jmaiel, J. Gómez-Montalvo, E. Exposito. "Towards a semantic-driven and scalable publish/subscribe framework", International Journal of Internet Protocol Technology, Inderscience Journals, 2013.

Conférences Internationales

– A. Chaabane, F. Abdennadher, W. Louati and M. Jmaiel. "Handling Churn in DHT-based Publish/Subscribe Systems", in Proceedings of the 3rd IEEE/IFIP International Conference on the Network of the Future (NoF 2012), Tunis, Tunisia, November 2012.

– A. Chaabane, W. Louati, M. Jmaiel, J. R. Gomez-Montalvo, C. Diop and E. Exposito. "Towards an Ontology and DHT-based publish/subscribe scalable system", in Proceedings of the 3rd IEEE International Workshop on SmArt COmmunications in NEtwork Technologies (ICC'12 WS - SaCoNet-III), Ottawa, Canada, June 2012.

– A. Chaabane, W. Louati and M. Jmaiel. "A Framework for Managing Composed Multimedia Delivery in Personal Networks", in Proceedings of the 2nd International Conference on Multimedia Computing and Systems (ICMCS'11), Ouarzazate, Morocco, April 2011. IEEE digital library.

Bibliographie

[1] G. Pallis, D. Zeinalipour-Yazti, and M. D. Dikaiakos, "Online social networks : Status and trends," in *New Directions in Web Data Management 1*, 2011, pp. 213–234.

[2] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta, "Peerson : P2p social networking : Early experiences and insights," in *Proceedings of the 2nd ACM EuroSys Workshop on Social Network Systems*. ACM, 2009, pp. 46–52.

[3] L. A. Cutillo, R. Molva, and M. Önen, "Safebook : A distributed privacy preserving online social network," in *Proceedings of the 12th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WOWMOM*, 2011, pp. 1–3.

[4] S. Seong, J. Seo, M. Nasielski, D. Sengupta, S. Hangal, S. K. Teh, R. Chu, B. Dodson, and M. S. Lam, "Prpl : A decentralized social networking infrastructure," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing &Services : Social Networks and Beyond*, ser. MCS '10. ACM, 2010, pp. 1–8.

[5] R. Sharma and A. Datta, "Supernova : Super-peers based architecture for decentralized online social networks," *CoRR*, vol. abs/1105.0074, 2011.

[6] A. Shakimov, H. Lim, R. Cáceres, O. P. Cox, K. Li, D. Liu, and E. Varshavsky, "Vis-á-vis : Privacy-preserving online social networking via virtual individual servers," in *In COMSNETS*, 2011.

[7] S. Nilizadeh, S. Jahid, P. Mittal, N. Borisov, and A. Kapadia, "Cachet : A decentralized architecture for privacy preserving social networking with caching," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '12. ACM, 2012, pp. 337–348.

[8] D. Sandler and D. S. Wallach, "Birds of a FETHR : open, decentralized micropublishing," in *Proceedings of the 8th international conference on Peer-to-peer systems*, 2009, p. 1.

[9] T. Xu, Y. Chen, J. Zhao, and X. Fu, "Cuckoo : Towards decentralized, socio-aware online microblogging services and data measurements," in *Proceedings of the 2nd ACM International Workshop on Hot Topics in Planet-scale Measurement*, ser. HotPlanet '10. ACM, 2010, pp. 1–4.

[10] M. Ripeanu, "Peer-to-peer architecture case study : Gnutella network," in *Proceedings of the 1st International Conference on Peer-to-Peer Computing*, 2001, pp. 99–100.

[11] A. Chaabane, W. Louati, and M. Jmaiel, "A framework for managing composed multimedia delivery in personal networks," in *Proceedings of the 2nd International Conference on Multimedia Computing and Systems, ICMCS*. IEEE digital library, 2011, pp. 1–6.

[12] O. K. Sahingoz and N. Erdogan, "Rubces : Rule based composite event system," *in XII. Turkish Artificial Intelligence and Neural Network Symposium (TAINN)*, 2003.

[13] P. R. Pietzuch, B. Shand, and J. Bacon, "A framework for event composition in distributed systems," in *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, ser. Middleware '03. Springer-Verlag New York, Inc., 2003, pp. 62–82.

[14] M. Gao, X. Yang, R. Jain, and B. C. Ooi, "Spatio-temporal event stream processing in multimedia communication systems," in *Proceedings of the 22nd International Conference, Scientific and Statistical Database Management, SSDBM*, 2010, pp. 602–620.

[15] S. Lai, J. Cao, and Y. Zheng, "Psware : a publish / subscribe middleware supporting composite event in wireless sensor network," in *Proceedings of the 7th Annual IEEE International Conference on Pervasive Computing and Communications - Workshops (PerCom Workshops 2009)*, 2009, pp. 1–6.

[16] S. Courtenage and S. Williams, "The design and implementation of a p2p-based composite event notification system," in *Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA*, 2006, pp. 701–706.

[17] M. J. J. Reagle and G. Turner, "User-generated content," *New Media And Society*, vol. 14, no. 7, pp. 1236–1239, 2012.

[18] C. W. Paper, "Cisco visual networking index : Global mobile data traffic forecast update, 2015-2020 white paper," Cisco, Tech. Rep., 2015.

[19] A. I. T. Rowstron and P. Druschel, "Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proceedings of the IFIP/ACM In-*

ternational Conference on Distributed Systems Platforms Heidelberg. Springer-Verlag, 2001, pp. 329–350.

[20] M. Castro, P. Druschel, A. M. Kermarrec, and A. I. Rowstron, "Scribe : A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1489–1499, 2006.

[21] F. Michel, "Définir et analyser les réseaux sociaux," *Informations sociales*, vol. 3, no. 147, p. 138, 2008.

[22] D. M. Boyd and N. B. Ellison, "Social network sites : Definition, history, and scholarship," *Journal of Computer-Mediated Communication*, vol. 13, no. 1, 2007.

[23] Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong, "Analysis of topological characteristics of huge online social networking services," in *Proceedings of the 16th International Conference on World Wide Web*. ACM, 2007, pp. 835–844.

[24] W. Willinger, R. Rejaie, M. Torkjazi, M. Valafar, and M. Maggioni, "Research on online social networks : Time to face the real challenges," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 3, pp. 49–54, 2010.

[25] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin, "Persona : An online social network with user-defined privacy," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 135–146, 2009.

[26] B. Carminati, E. Ferrari, and A. Perego, "Enforcing access control in web-based social networks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, pp. 1–6, 2009.

[27] P. W. L. Fong, M. Anwar, and Z. Zhao, "A privacy preservation model for facebook-style social network systems," in *Proceedings of the 14th European Conference on Research in Computer Security*. Springer-Verlag, 2009, pp. 303–320.

[28] A. Tootoonchian, K. K. Gollu, S. Saroiu, Y. Ganjali, and A. Wolman, "Lockr : Social access control for web 2.0," in *Proceedings of the 1st Workshop on Online Social Networks*. ACM, 2008, pp. 43–48.

[29] W. Villegas, B. Ali, and M. Maheswaran, "An access control scheme for protecting personal data," in *Proceedings of the 2008 6th Annual Conference on Privacy, Security and Trust*. IEEE Computer Society, 2008, pp. 24–35.

[30] R. Buyya, M. Pathan, and A. Vakali, *Content Delivery Networks*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[31] G. Pallis and A. Vakali, "Insight and perspectives for content delivery networks," *Commun. ACM*, vol. 49, no. 1, pp. 101–106, 2006.

[32] A. Shakimov, A. Varshavsky, L. P. Cox, and R. Cáceres, "Privacy, cost, and availability tradeoffs in decentralized osns," in *Proceedings of the 2nd ACM Workshop on Online Social Networks*. ACM, 2009, pp. 13–18.

[33] S. Buchegger and A. Datta, "A case for p2p infrastructure for social networks - opportunities & challenges," in *Proceedings of the 6th International Conference on Wireless On-Demand Network Systems and Services*. IEEE Press, 2009, pp. 149–156.

[34] B. Carminati, E. Ferrari, and A. Perego, "Enforcing access control in web-based social networks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, pp. 1–6, 2009.

[35] A. Nazir, S. Raza, D. Gupta, C.-N. Chuah, and B. Krishnamurthy, "Network level footprints of facebook applications," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*. ACM, 2009, pp. 63–75.

[36] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger, "Understanding online social network usage from a network perspective," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*. ACM, 2009, pp. 35–48.

[37] W. Willinger, R. Rejaie, M. Torkjazi, M. Valafar, and M. Maggioni, "Research on online social networks : time to face the real challenges," *SIGMETRICS Performance Evaluation Review*, vol. 37, no. 3, pp. 49–54, 2009.

[38] S. Buchegger and A. Datta, "A case for p2p infrastructure for social networks - opportunities & challenges," in *Proceedings of the 6th International Conference on Wireless On-Demand Network Systems and Services*, ser. WONS'09. IEEE Press, 2009, pp. 149–156.

[39] D. Recordon and D. Reed, "Openid 2.0 : A platform for user-centric identity management," in *Proceedings of the 2nd ACM Workshop on Digital Identity Management*, ser. DIM '06. ACM, 2006, pp. 11–16.

[40] M. K. Denko, E. M. Shakshuki, and H. Malik, "A mobility-aware and cross-layer based middleware for mobile ad hoc networks," in *Proceedings of the 21st International Confe-*

rence on Advanced Information Networking and Applications (AINA 2007), 2007, pp. 474–481.

[41] S. Jahid, S. Nilizadeh, P. Mittal, N. Borisov, and A. Kapadia, "DECENT : A decentralized architecture for enforcing privacy in online social networks," in *Proceedings of the 10th Annual IEEE International Conference on Pervasive Computing and Communications*, 2012, pp. 326–332.

[42] N. S. Good and A. Krekelberg, "Usability and privacy : A study of kazaa p2p file-sharing," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '03. ACM, 2003, pp. 137–144.

[43] C. Shirky, "What is p2p and what isn't," in *Proceedings of the O'Reilly Peer to Peer and Web Service Conference*, 2001.

[44] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Computing Surveys (CSUR)*, vol. 36, no. 4, pp. 335–371, 2004.

[45] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet : A distributed anonymous information storage and retrieval system," in *Proceedings of the International Workshop On Designing Privacy Enhancing Technologies : Design Issues In Anonymity and Unobservability*. Springer-Verlag New York, Inc., 2001, pp. 46–66.

[46] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord : A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, 2001, pp. 149–160.

[47] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 161–172, 2001.

[48] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt, "P-grid : A self-organizing structured p2p system," *SIGMOD Rec.*, vol. 32, no. 3, pp. 29–33, 2003.

[49] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry : A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, pp. 41–53, 2004.

[50] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Information and System Security (TISSEC)*, vol. 19, no. 3, pp. 332–383, 2001.

[51] G. Mühl, L. Fiege, and P. R. Pietzuch, *Distributed event-based systems*. Springer, 2006.

[52] Y. Liu and B. Plale, "Survey of publish subscribe event systems," 2003.

[53] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Interfaces and algorithms for a wide-area event notification service," 2000.

[54] W. Louati and D. Zeghlache, "Personal overlay networks management using a p2p-based publish/subscribe naming system," in *Proceedings of the 16th Euromicro International Conference on Parallel, Distributed and Network-Based*, 2008, pp. 95–99.

[55] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz, "Bayeux : An architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM, 2001, pp. 11–20.

[56] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps, "Content based routing with elvin4," in *Proceedings of AUUG2K*, 2000.

[57] P. R. Pietzuch and J. Bacon, "Hermes : A distributed event-based middleware architecture," in *Proceedings of the 22nd International Conference on Distributed Computing Systems*, ser. ICDCSW '02, 2002, pp. 611–618.

[58] I. Aekaterinidis and P. Triantafillou, "Pastrystrings : A comprehensive content-based publish/subscribe dht network," in *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, ser. ICDCS '06), 2006, p. 23.

[59] D. K. Tam, R. Azimi, and H. Jacobsen, "Building content-based publish/subscribe systems with distributed hash tables," in *Proceedings of the 1st International Workshop Databases, Information Systems, and Peer-to-Peer Computing, DBISP2P, Revised Selected Papers*, 2003, pp. 138–152.

[60] R. V. Renesse and A. Bozdog, "Willow : Dht, aggregation, and publish/subscribe in one protocol," in *Proceedings of the 3rd International Workshop Peer-to-Peer Systems III, IPTPS, Revised Selected Papers*, 2004, pp. 173–183.

[61] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi, "Meghdoot : Content-based publish/subscribe over p2p networks," in *Proceedings of the 5th ACM/IFIP/USENIX*

International Conference on Middleware, ser. Middleware '04. Springer-Verlag New York, Inc., 2004, pp. 254–273.

[62] N. Hernandez, "Ontologies de domaine pour la modélisation du contexte en recherche d'information," Ph.D. dissertation, Université Paul Sabatier - Toulouse III, 2006.

[63] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proceedings of ACM SIGCOMM 2001*, 2001, pp. 161–172.

[64] J. Keeney, D. Roblek, D. Jones, D. Lewis, and D. O'Sullivan, "Extending siena to support more expressive and flexible subscriptions," in *Proceedings of the 2nd International Conference on Distributed Event-based Systems*, ser. DEBS'08. ACM, 2008, pp. 35–46.

[65] J. Skovronski and K. Chiu, "Ontology based publish subscribe framework," in *Proceedings of the 8th International Conference on Information Integration and Web-based Applications Services*, ser. iiWAS'2006, 2006, pp. 49–58.

[66] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. R. Swartout, "Enabling technology for knowledge sharing," *AI Magazine*, vol. 12, no. 3, pp. 36–56, 1991.

[67] N. Aussenac-Gilles, B. Biébow, and S. Szulman, "Modélisation du domaine par une méthode fondée sur l'analyse de corpus," in *Ingénierie des Connaissances*, R. Teulier, J. Charlet, and P. Tchounikine, Eds. L'Harmattan, 2005, pp. 40–72.

[68] M. M. Taye, "Understanding semantic web and ontologies : Theory and applications," *CoRR*, vol. abs/1006.4567, 2010.

[69] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, "Swrl : A semantic web rule language combining owl and ruleml," in *W3c member submission, World Wide Web Consortium*, 2004.

[70] A. Chaabane, W. Louati, M. Jmaiel, J. R. Gomez-Montalvo, C. Diop, and E. Exposito, "Towards an Ontology and DHT-based publish/subscribe scalable system," in *Proceedings of IEEE International Conference on Communications, ICC 2012*, 2012, pp. 6499–6503.

[71] A. Mallik, P. Pasumarthi, and S. Chaudhury, "Multimedia ontology learning for automatic annotation and video browsing," in *Proceedings of the 1st ACM International Conference on Multimedia Information Retrieval*, ser. MIR '08. ACM, 2008, pp. 387–394.

[72] A. Chaabane, C. Diop, W. Louati, M. Jmaiel, J. Gomez-Montalvo, and E. Exposito, "Towards a semantic-driven and scalable publish/subscribe framework," *International Journal of Internet Protocol Technology*, vol. 7, no. 3, pp. 165–175, 2013.

[73] D. L. N. Alliance, "Dlna âĂŞ overview and vision, white paper," DLNA, Tech. Rep., 2007.

[74] R. Tusch, M. Jakab, J. KÃűpke, A. KrÃďtschmer, M. Kropfberger, S. Kuchler, M. Ofner, H. Hellwagner, and L. BÃűszÃűrmenyi, "Context-aware upnp-av services for adaptive home multimedia systems," *International Journal of Digital Multimedia Broadcasting*, vol. Vol. 2008, p. 12, 2008.

[75] A. Y. kyoung Song, A. Bhagwat, B. Fairman, D. Hlasny, D. Verslype, F. Tuck, J. Nelis, M. van Hartskamp, N. Gadiraju, P. Sharma, R. Chen, S. Gavette, S. Wade, S. Sharma, and Z. Wu, "Upnp-qos architecture : 3 for upnp version 1.0 status : Standardized dcp," 2008.

[76] A. Presser, L. Farrell, D. Kemp, W. Lupton, S. Tsuruyama, S. Albright, A. Donoho, J. Ritchie, B. Roe, M. Walker, T. Nixon, C. Evans, H. Rawas, T. Freeman, J. Park, C. Chan, F. Reynolds, J. Costa-Requena, Y. Ye, T. McGee, G. Knapen, M. Bodlaender, J. Guidi, L. Heerink, J. Gildred, A. Messer, Y. Kim, M. Wischy, A. Fiddian-Green, B. Fairman, J. Tourzan, and J. Fuller, "Upnp™ device architecture 1.1," 2008.

[77] "The encompas (enabling community communications - platforms and applications) project website, http ://encompas.org/," 2007-2009.

[78] D. Zimmer and R. Unland, "The formal foundation of the semantics of complex events in active database management systems," Cooperative Computing and Communication Laboratory, Tech. Rep., 1997.

[79] D. C. Luckham, *The Power of Events : An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[80] A. Hinze and A. Buchmann, Eds., *Principles and Applications of Distributed Event-Based Systems*. IGI Global, 2010.

[81] S. Courtenage, "Specifying and detecting composite events in content-based publish/-subscribe systems," in *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops*, ser. ICDCSW '02, 2002.

[82] J. Qian, J. Yin, J. Dong, and D. Shi, "Jtangcsps : A composite and semantic publish/-subscribe system over structured p2p networks," *Eng. Appl. Artif. Intell.*, vol. 24, no. 8, pp. 1487–1498, 2011.

[83] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling churn in a dht," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '04. USENIX Association, 2004, pp. 10–23.

[84] M. Landers, H. Zhang, and K.-L. Tan, "Peerstore : Better performance by relaxing in peer-to-peer backup," in *Peer-to-Peer Computing*, 2004, pp. 72–79.

[85] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," in *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, ser. SOSP '01. ACM, 2001, pp. 202–215.

[86] B. Wong and S. Guha, "Quasar : A probabilistic publish-subscribe system for social networks," in *Proceedings of the 7th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2008.

[87] R. Narendula, T. G. Papaioannou, and K. Aberer, "Towards the realization of decentralized online social networks : An empirical study," in *Proceedings of the 32nd International Conference on Distributed Computing Systems Workshops (ICDCS 2012 Workshops)*, 2012, pp. 155–162.

[88] S. Forsyth and K. Daudjee, "Update management in decentralized social networks," in *Proceedings of the 33rd International Conference on Distributed Computing Systems Workshops (ICDCS 2013 Workshops)*, 2013, pp. 196–201.

[89] H. Asthana and I. J. Cox, "A framework for peer-to-peer micro-blogging," in *Proceedings of the 33rd International Conference on Distributed Computing Systems Workshops (ICDCS 2013 Workshops)*, 2013, pp. 184–189.

[90] R. Rodrigues and C. Blake, "When multi-hop peer-to-peer lookup matters," in *Proceedings of the 3rd International Workshop, Peer-to-Peer Systems III, IPTPS, Revised Selected Papers*, 2004, pp. 112–122.

[91] M. Castro, M. Costa, and A. I. T. Rowstron, "Performance and dependability of structured peer-to-peer overlays," in *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, 2004, pp. 9–18.

[92] F. Dabek, J. Li, E. Sit, J. Robertson, M. F. Kaashoek, and R. Morris, "Designing a DHT for low latency and high throughput," in *Proceedings of the 1st Symposium on Networked Systems Design and Implementation NSDI*, 2004, pp. 85–98.

[93] A. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," *SIGOPS Oper. Syst. Rev.*, vol. 35, no. 5, pp. 188–201, 2001.

[94] A. Chaabane, F. Abdennadher, W. Louati, and M. Jmaiel, "Handling Chrun in DHT-based Publish/Subscribe Systems," in *Proceedings of the 3rd IEEE/IFIP International Conference on the Network of the Future (NoF 12)*. IEEE Communications Society, 2012.

[95] F. Trichet, X. Aimé, and C. Thovex, *Handbook of Research on Culturally-Aware Information Technology : Perspectives and Models (1 Volume)*. Information Science Reference - Imprint of : IGI Publishing, 2010, ch. OSIRIS : Ontology-based System for Semantic Information Retrieval and Indexation dedicated to community and open web spaces.