Exploration de l'espace des configurations architecturales guidé par la QdS des systèmes Publier/Souscrire sur MANET

Rania Abid, Imene Lahyani, Ismael Bouassida Rodriguez, Mohamed Jmaiel

Exploration de l'espace des configurations architecturales guidé par la QdS des systèmes Publier/Souscrire sur MANET

Rania Abid
Laboratoire ReDCAD, Enis,
Université de Sfax, Tunisia
raniaenis@gmail.com

Imene Lahyani
CNRS, LAAS, 7 avenue du
colonel Roche, F-31400
Toulouse, France
Univ de Toulouse, LAAS,
F-31400 Toulouse, France
ReDCAD, University of Sfax,
B.P. 1173, 3038 Sfax, Tunisia
iabdenna@laas.fr

Ismael Bouassida Rodriguez
CNRS, LAAS, 7 avenue du
colonel Roche, F-31400
Toulouse, France
Univ de Toulouse, LAAS,
F-31400 Toulouse, France
ReDCAD, University of Sfax,
B.P. 1173, 3038 Sfax, Tunisia
bouassida@laas.fr

Mohamed Jmaiel
Laboratoire ReDCAD, Enis,
Université de Sfax, Tunisia
mohamed.jmaiel@enis.rnu.tn

Résumé- Une préoccupation importante dans le domaine des systèmes distribués est d'améliorer la propagation de l'information et plus particulièrement dans les systèmes Publier/Souscrire déployés sur les réseaux mobiles ad hoc (MANET). En fait, la réalisation de ce but se confronte à différents problèmes tels que la défaillance du système de communication et la dégradation de la qualité de service (QdS). Nous nous focalisons sur le traitement du problème de la QdS en abordant les solutions possibles pour assurer un fonctionnement performant et fiable de ces systèmes. Nous proposons pour cette finalité des actions de reconfiguration applicables sur un réseau présentant une dégradation de qualité de service en utilisant des graphes pour la modélisation de ce système distribué et un moteur d'appariement et de transformation de graphes (GMTE) pour exécuter les règles de reconfigurations. La solution génerée est sélectionnée selon différents critères d'évaluation afin de répondre aux exigences du système en termes de QdS.

1. INTRODUCTION

Un système Publier/Souscrire, également connu sous le nom de système basé événements, a été adopté comme une technologie clé pour la diffusion efficace de l'information sur une grande échelle. Ce système est constitué d'un ensemble de noeuds distribués jouant le rôle d'un service d'évènements, des producteurs et des consommateurs. Le service d'événement est un ensemble de *dispatchers* responsables de la distribution et de l'acheminement des notifications du producteur aux consommateurs concernés à travers des liens que nous appelons liens logiques. Cet ensemble peut être appelé niveau logique. Ce système est très évolutif car il prend en charge un grand nombre de participants (producteurs et consommateurs) et se caractérise par un découplage dans le temps, dans l'espace et de synchronisation. Par ailleurs, nous nous situons dans le contexte de réseau MANET, qui est formée par un grand nombre d'unités/équipements mobiles communiquant par l'intermédiaire d'interfaces sans fil. Ce réseau se distingue par l'absence d'une infrastructure et par la facilité et la rapidité de déploiement des entités mobiles en établissant des échanges directs. En outre, ce réseau se caractérise par une gestion décentralisée car elle est répartie sur les différentes entités mobiles. La contrainte d'énergie est très importante car les noeuds sont alimentés par des batteries à durée de vie limitée. Par ailleurs la bande passante est limitée car le support est sans fil. Dans ce contexte de systèmes distribués, nous cherchons à déployer une architecture Publier/Souscrire sur un réseau MANET. Les noeuds du réseau MANET se caractérise notamment par la mobilité et l'autonomie. En fait, cette mobilité provoque la dégradation de la performance du lien logique. En outre, l'autonomie des noeuds affecte la performance des entités logiques (*dispatchers*, producteurs et consommateurs). Cette influence sur les performances du réseau logique (entités et liens logiques) entraine une dégradation des paramètres de la qualité de service (QdS) au niveau logique. Il est donc impératif de maintenir une bonne QdS et de réparer la dégradation avec des techniques appropriées.

Dans cet article, nous proposons une approche qui consiste à développer/élaborer des techniques permettant la détection de la dégradation de la QdS et la réparation en appliquant des politiques de reconfiguration (que nous avons développées). Cette approche permet l'évaluation de ces politiques et la sélection du déploiement qui améliore l'état du réseau.

Nous abordons dans la section 2 les travaux connexes. Puis, nous présentons notre problématique et le cas d'étude dans la section 3. La section 4 est consacrée pour la QdS. Puis nous détaillons notre approche dans la section 5. Le reste de l'article sera divisé entre la section 6, qui présente l'implémentation, la section 7 qui est consacrée à l'application de notre approche sur notre cas d'étude et une conclusion dans la section 8.

2. TRAVAUX CONNEXES

Après un balayage des travaux proposant des actions de réparation et des méthodes d'évaluation des architectures, nous avons trouvé que du côté middleware, la majorité des travaux existants permet une amélioration partielle de la QdS et peut introduire d'autres problèmes qui affectent l'état général du système Publier/Souscrire déployé sur un réseau MANET. La majorité de ces travaux se répartissent sur deux catégories. La première catégorie concerne les systèmes d'autoréparation qui cherchent à maintenir la connectivité entre les *dispatchers* au niveau middleware en recourant à la réparation du noeud ou du lien. Parmi ces systèmes il est possible de citer REDS [5] et Pusman [4] qui se basent sur la substitution du *dispatcher* défaillant par un autre ce qui réduit sensiblement le nombre des *dispatchers*. Aussi, la panne d'un point d'accès ne peut pas être réparée. La deuxième catégorie cherche à garantir les performances en satisfaisant les exigences en quelques paramètres de QdS. Ces systèmes recourent à une amélioration périodique pour assurer quelques exigences de QdS. Parmi ces systèmes se citent IndiQoS [3], Harmony [7] et Système Q [1]. Le principe d'IndiQos est d'introduire des attributs de QdS dans les souscriptions. Il utilise une approche de réservation de ressources pour contrôler la latence. Toutefois cette inclusion peut empêcher l'acheminement de notification si elle ne vérifie pas ces attributs de QdS. En outre, IndiQos s'appuie sur les messages d'alerte ce qui engendre une perte de temps et de ressources surtout dans MANET. Harmony utilise la technique "proactive best-path" pour déterminer la route la plus courte menant à chaque destination en se basant sur un agent monitoring qui mesure les latences. Mais, cette technique ne permet pas de déterminer si la route actuelle maintient l'exigence de QdS spécifiée par l'application. Ce dernier utilise aussi une technique "Multipath" qui retransmet les évènements sur deux routes pour augmenter la fiabilité. Mais, cette technique peut engendrer la congestion du réseau suite à la redondance des informations. Le système Q applique des opérations de substitution de liens logiques et utilise une technique de comparaison des distances séparant les noeuds logiques pour optimiser périodiquement les routes afin de minimiser la latence. Toutefois, la substitution du lien s'avère parfois inutile si avec une distance plus minimale la latence va augmenter et non pas diminuer. Comme déjà mentionné, ces systèmes assurent localement l'exigence en QdS mais il n'y a aucune garantie pour le réseau global. Ainsi, ces systèmes sont incapables de satisfaire notre objectif qui consiste à garantir un bon fonctionnement d'un système Publier/Souscrire sur MANET en maintenant la QdS exigée lors de la transmission des messages. Concernant les travaux d'évaluation d'architecture, nous avons trouvé qu'ils se basent sur différents critères d'évaluation et différents types d'attributs de QdS. En effet, il y a ceux qui évaluent les architectures en se basant sur les exigences en QdS et ceux qui évaluent les architectures en

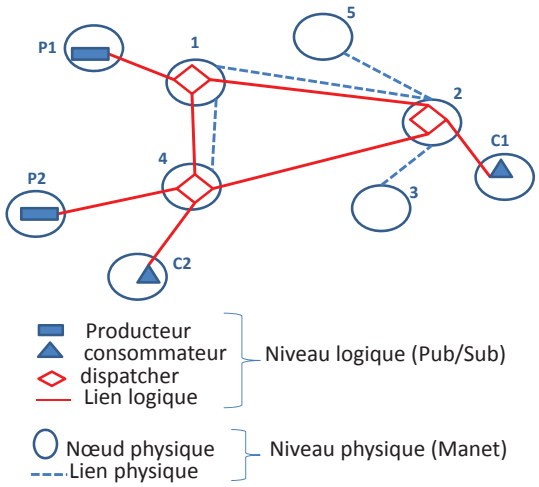

Figure 1: Exemple de déploiement d'un système Publier/Souscrire sur MANET

se basent sur le contexte de l'application. Pour les travaux d'évaluation d'architecture qui se basant sur les exigences en QdS, ils répartissent les attributs de QdS en deux types[6]. Le premier type concerne les attributs de QdS temps réel du système à savoir la performance, la sécurité et le deuxième type concerne les attributs de QdS de conception qui ne peuvent être évalués que tout au long du cycle de vie de l'application à savoir la maintenabilité et la flexibilité. En fait, il y a des politiques d'évaluation qui se concentrent sur un attribut particulier[6] de QdS tels que la "Performance" ou la "Sécurité", par exemple, à quelle mesure l'architecture a le potentiel de soutenir un certain nombre de transactions par seconde, ou si une architecture soutiendra un temps spécifié. Il existe également différentes méthodes qui sont disponibles dans l'industrie pour évaluer les architectures, fondés exclusivement sur des facteurs tels que le coût, la modificabilité et l'interopérabilité selon le contexte de l'application[6]. Par exemple, le nombre des lignes de code est considéré comme un attribut de système qui peut être utilisé pour évaluer la propriété volume.

Pour notre travail, nous avons créé nos propres critères d'évaluation selon la particularité de l'architecture Publier/-Souscrire déployé sur MANET que nous étudions. Nous nous concentrons sur la réparation des dégradations détectés dans ce type d'architecture en visant un nombre minimum de problèmes au niveau des entités et des liens sur le réseau.

3. PROBLÉMATIQUE ET CAS D'ÉTUDE

Le déploiement d'une architecture Publier/Souscrire sur un réseau MANET se confronte à des problèmes de QdS liés aux caractéristiques des réseaux MANET notamment :

- **La mobilité** des noeuds influent sur les liens physiques entre les équipements. Ceci engendre la dégradation des performances **des liens logiques** entre les entités logiques.
- **L'autonomie** des noeuds se traduit par des ressources limitées en termes d'énergie, de mémoire et de capacité de traitement. Ceci influe sur les performances **des entités logiques**.

Cette influence sur les performances du réseau logique se traduit par une **dégradation des paramètres de QdS**. Ainsi notre but est d'assurer le bon fonctionnement des systèmes Publier/Souscrire déployés sur MANET tout en maintenant la QdS au niveau de la transmission des messages. De ce fait, notre solution a pour but de rapprocher la QdS offerte par l'architecture Publier/Souscrire de celle exigée par l'utilisateur de MANET et de la maintenir au niveau des liens et des entités logiques.

Nous prenons pour notre cas d'étude l'exemple (Figure 1) de déploiement d'un système Publier/Souscrire sur MANET formé par un niveau logique qui est constitué de deux producteurs, un consommateur et un service d'évènements formé de trois *dispatchers* et un niveau physique qui est formé par des noeuds physiques. Entre deux noeuds il existe au moins un chemin physique.

4. ETUDE DES PARAMÈTRES DE QDS

Dans cette section, nous étudions les paramètres de QdS pour les systèmes Publier/Souscrire déployés sur MANET.

4.1 Pour un dispatcher

Dans ce paragraphe, nous étudions les paramètres de QdS requis pour un bon fonctionnement d'un *dispatcher* déployé sur un noeud. Pour cette entité de l'architecture Publier/-Souscrire, nous cherchons à assurer une longue durée de vie du *dispatcher*. Donc nous avons besoin de la mémoire vive et de l'énergie de la machine sur laquelle il est déployé. Et nous cherchons à assurer aussi que la machine ne soit pas chargée par plusieurs applications. Donc nous avons besoin de prendre en considération le temps d'exécution (la charge de CPU : le taux d'utilisation de CPU). Ainsi, les paramètres de QdS à étudier sont **l'énergie**, **la RAM** et **la charge CPU**.

4.2 Pour un lien logique

Dans ce paragraphe, nous étudions les paramètres de QdS requis pour un transfert réussi de messages entre les *dispatchers* à travers un lien logique. Nous cherchons à assurer l'arrivée du message de la source à la destination à temps (délai prédéfini appartenant à un intervalle précis) avec un taux de perte qui tend vers 0. Par ailleur, il faut assurer que la charge de chaque lien logique n'empêche pas de transférer tous les messages envoyés entre les deux *dispatchers* extrémités du lien. Ainsi, les paramètres de QdS à étudier sont **le taux de perte**, **la latence** et **la charge du lien logique**. Nous définissons le délai (latence) comme le temps d'acheminement d'un évènement d'un *dispatcher* à un autre voisin. Afin de remédier aux dégradations des paramètres de QdS, nous proposons d'appliquer des actions de réparation (reconfiguration) convenables à chaque cause de dégradation de QdS (le paramètre dégradé).

5. APPROCHE

Notre approche se situe dans le cadre d'un processus de reconfiguration des systèmes Publier/Souscrire sur MANET.

Nous intervenons dans l'étape "Analyse" qui consiste à détecter la dégradation de QdS et à identifier l'état du système, puis dans l'étape "Planification" qui consiste à déterminer l'action de reconfiguration convenable et enfin dans l'étape "Exécution" qui consiste à exécuter l'action de reconfiguration convenable.

L'idée de l'approche est de tracer l'arbre des solutions possibles des actions de reconfiguration pouvant être appliquées dans un cas de dégradation et de sélectionner une configuration du réseau réparé selon des critères d'évaluation. Cette approche se concentre sur le niveau des modèles en appliquant la reconfiguration sur le réseau dégradé (décrit sous forme de modèle). Les étapes d'application de la réparation au niveau modèle reviennent en premier lieu à la description du réseau dégradé par graphe. Puis à la description de l'action de reconfiguration par règle de réécriture de graphe et de l'appliquer sur le graphe avec le moteur **GMTE**[1]. Après, l'exécution de l'action de réparation nous évaluons les graphes résultants et nous sélectionnons un graphe de sortie.

5.1 Actions de reconfigurations proposées

Nous définissons une reconfiguration comme étant un processus basé règles conduisant à un changement de la topologie des *dispatchers*, elle comprend le retrait/l'ajout d'un *dispatcher* et la substitution d'un lien.

5.1.1 La migration d'un dispatcher

La migration d'un *dispatcher* est applicable en cas de dégradation détectée au niveau de la machine sur laquelle un *dispatcher* est déployé (au niveau de la RAM, CPU ou énergie) ou bien en cas de dégradation au niveau d'un lien logique. Cette action suit trois étapes :

1. **Initialisation :** Choisir un noeud physique du réseau qui assure une meilleure QdS.
2. **Transfert des données :** Envoi de la table de souscription au nouveau *dispatcher*.
3. **Informer les voisins de la nouvelle localisation,** le *dispatcher* défaillant s'arrête et le nouveau *dispatcher* commence à fonctionner au même temps.

5.1.2 Substitution d'un lien logique

La substitution est applicable en cas de dégradation détectée au niveau d'un lien logique reliant deux *dispatchers* (au niveau de la latence, taux de perte, charge du lien logique). La substitution d'un lien logique défaillant revient à chercher à intercaler un *dispatcher* intermédiaire entre les deux *dispatchers* extrémités du lien logique.

5.1.3 Eclatement d'un dispatcher

L'éclatement d'un *dispatcher* en deux est applicable en cas de dégradation de QdS au niveau de la machine sur laquelle le *dispatcher* est déployé. Elle suit le même principe que la migration sauf que le *dispatcher* concerné sera éclaté en deux nouveaux *dispatchers* lancés sur deux machines destinataires.

5.2 Modélisation de l'architecture

Dans cette sous-section, nous présentons les éléments de modélisation d'un graphe Publier/Souscrire déployé sur MANET que nous appliquons sur note cas d'étude.

1. http ://homepages.laas.fr/khalil/GMTE

5.2.1 *Eléments de modélisation du graphe*

Nous modélisons l'architecture par un graphe [2] formé par deux types de sommets et deux types d'arcs qui sont caractérisés par :

- **2 types de sommets :**
  - $D_{Dx}(ID_{Dx},IP_x)$: représente un *dispatcher* ayant un identifiant ID_{Dx} et déployé sur une machine ayant une adresse ip IP_x.
  - $N_x(ID_x, IP_x, RAM_x, W_x, CPU_x)$: représente une machine ayant un identifiant ID_x et une adresse IP_x et les pourcentages restants en RAM (RAM_x), en énergie(W_x) et en pourcentage d'utilisation de CPU (CPU_x).
- **2 types d'arcs :**
  - **Type1 :** "Deployed on" (figure 2) : indiquant qu'un *dispatcher* D_{Dx} est déployé sur un noeud N_x.
  - **Type2 :** (figure 3) Arc indiquant le lien (physique/logique) entre deux noeuds : type physique, type logique, bande passante logique, latence, taux de perte, charge du lien.

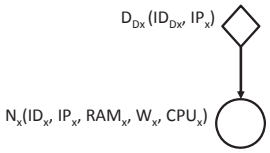

Figure 2: Le *dispatcher* D_{Dx} est déployé sur le noeud N_x

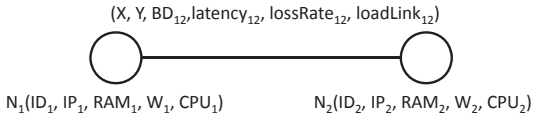

Figure 3: Lien entre les noeuds N_1 et N_2

Le lien (X, Y, BD_{12}, $latency_{12}$, $lossRate_{12}$, $loadLink_{12}$) (figure 3) peut être :

- (P,-, BD_{12}, $latency_{12}$, $lossRate_{12}$, $loadLink_{12}$) : un saut physique entre les deux noeuds.
- (-, L, BD_{12}, $latency_{12}$, $lossRate_{12}$, $loadLink_{12}$) : un lien logique moyennant n sauts physiques.
- (P, L, BD_{12}, $latency_{12}$, $lossRate_{12}$, $loadLink_{12}$) : un lien logique moyennant un saut physique.

5.2.2 *Modélisation du cas d'étude*

La modélisation du cas d'étude (décrit dans la section 4) à un instant Ti en graphe correspond au graphe d'architecture de la figure 4 :

Ce graphe est formé par les *dispatchers* D_{D1}, D_{D2} et D_{D4} qui sont déployés respectivement sur les noeuds N_1, N_2 et N_4. Sur ce graphe nous testons les différentes politiques de reconfiguration de notre approche. Nous définissons l'état d'un graphe à un instant Ti par l'expression suivante :

$$G(Problème\_Noeud/Problème\_Noeud)$$

avec

- $Problème\_Noeud = (nbr\_N(G) * 1, nbr\_A\_N)$

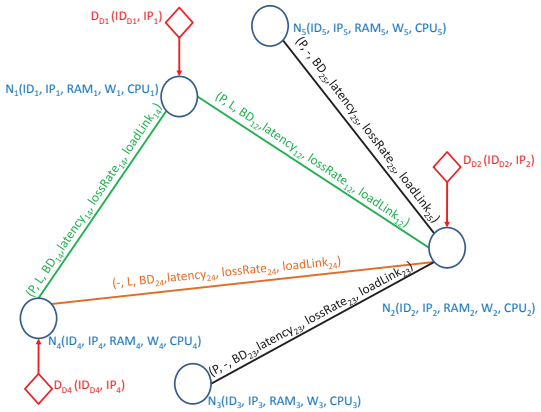

Figure 4: Graphe d'architecture Initale (AG)

- $Problème\_Lien = ((nbr\_L1(G) * 1, nbr\_A\_L1), (nbr\_L2(G) * 2, nbr\_A\_L2), (nbr\_L3(G) * 2, nbr\_A\_L3)))$

Avec :

- nbr_N(G) : le nombre de noeuds dégradés dans G et sur-lesquels sont déployés des *dispatchers*
- nbr_A_N : le nombre d' actions de reconfiguration appliquées sur le graphe pour réparer le Prob_Noeud
- nbr_L1(G) : le nombre de liens logiques dégradés dans G souffrant d'une dégradation (Mobility, Load ou Others)
- nbr_A_L1 : le nombre d'actions de reconfigurations appliquées sur le graphe pour réparer les liens souffrant d'une dégradation
- nbr_L2(G) : le nombre de liens logiques dégradés dans G souffrant de deux dégradations parmi (Mobility, Load, Others)
- nbr_A_L2 : le nombre d' actions de reconfigurations appliquées sur le graphe pour réparer les liens souffrant de deux dégradations
- nbr_L3(G) : le nombre de liens logiques dégradés dans G souffrant de trois dégradations (Mobility, Load et Others)
- nbr_A_L3 : le nombre d'actions de reconfigurations appliquées sur le graphe pour réparer les liens souffrant de trois dégradations

5.3 Mise en oeuvre du processus de reconfiguration

5.3.1 *Étape 1 : intervention dans la phase "Analyse" : Proposition de détection de dégradation*

La phase "Analyse" consiste à détecter la dégradation de QdS et à identifier l'état du système.

Nous proposons ainsi des formules de détection de dégradation au niveau des équipements contenant des *dispatchers* et au niveau des liens logiques. La dégradation d'un équipement peut être au niveau de la mémoire vive (RAM), de l'énergie (W) ou de la charge CPU (CPU). Nous détectons une dégradation au niveau de la RAM si le pourentage de la RAM restante est inférieur ou égale à une valeur x. Et pour la dégradation au niveau de CPU, elle est détectée si le pourcentage de la charge de CPU restante est supérieur ou égale à une valeur y. Alors que la dégradation au niveau de l'W est détectée si le pourcentage de l'W restante est inférieur ou égale à une valeur z. Les valeurs x, y et z sont

déclarées par le concepteur dans un fichier et elles sont modifiables.

La dégradation d'un lien logique peut avoir comme origine la mobilité des équipements, la charge d'un lien ou autres causes inconnues.

5.3.2 *Étape 2 : intervention dans la phase "Planification" : Proposition de politique de reconfiguration*

Nous classifions les différents problèmes confrontés dans un réseau Publier/Souscrire déployé sur MANET en deux catégories (table 1) : Problème_Noeud et Problème_Lien.

Type de dégradation	**Paramétre dégradé (Phase d'analyse)**	**Action à appliquer (Phase de planification)**
Problème_Noeud	RAM	Migration ou Eclatement
	CPU	Migration ou Eclatement
	W	Migration ou Eclatement
Problème_Lien	Latence ou/et Taux de perte	Migration ou Eclatement
	Charge du lien	Substitution
	Autres	Substitution

Table 1: Classification des dégradation(s)/reconfiguration(s)

Pour le(s) Problème_Noeud, la reconfiguration revient à appliquer la Migration ou l'Eclatement du *dispatcher* déployé sur le noeud dégradé. Pour le(s) Problème_Lien, la reconfiguration revient à appliquer la Migration ou l'Eclatement (du *dispatcher* émetteur ou récepteur) ou la Substitution du lien logique. Un noeud qui est dégradé peut souffrir d'un, ou de deux ou de trois problèmes au niveau de la RAM ou de la charge CPU ou de l'énergie. En effet, si un ou plusieurs paramètres de RAM, CPU ou W sont dégradés, les actions de reconfiguration à appliquer sont les mêmes (Migration ou Eclatement). Par contre, un lien logique qui est dégradé peut présenter un ou plusieurs problèmes parmi la mobilité, la charge ou autres et à chaque dégradation nous associons une action spécifique.

5.3.3 *Étape trois : intervention dans la phase "Exécution" : Proposition de reconfiguration et d'évaluation*

La phase Exécution consiste à appliquer l'action de reconfiguration sur le graphe d'entrée. Suite à cette exécution plusieurs alternatives de graphes réparés. Parmi ces graphes nous choisissons un graphe de sortie selon des critères d'évaluation. Nous proposons comme critères d'évaluation de choisir le graphe ayant :

- Un nombre minimal des équipements dégradés
- Un nombre minimal des liens logiques dégradés
- Un nombre minimal de(s) action(s) appliquée(s)

5.4 Présentation des règles de réécriture des graphes

Nous utilisons le moteur **GMTE** pour exécuter les règles de reconfiguration.Le **GMTE** est un moteur d'appariement et de transformation des graphes implémenté en C++. Il prend en entrée un graphe d'entrée et une règle et il donne en sortie un graphe de sortie. Le graphe d'entrée représente le réseau présentant une dégradation. La règle représente l'action de reconfiguration à appliquer. Le graphe de sortie représente le réseau après la réparation. Ces deux entrées sont appelés respectivement un graphe d'architecture et un graphe de règle.

5.4.1 *Migration de dispatcher*

La Migration de *dispatcher* est une action de reconfiguration que nous modélisons par une règle de réécriture des graphes. Cette règle est représentée comme le montre la figure 5 :

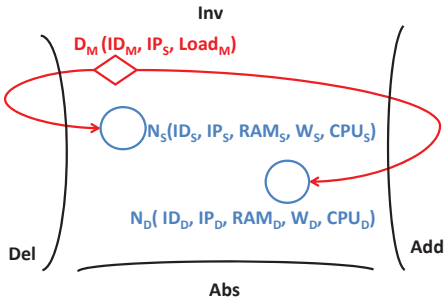

Figure 5: Règle de la Migration d'un *dispatcher*

Cette règle (figure 5) consiste à migrer un *dispatcher* D_M déployé sur un noeud dégradé N_S vers un noeud destinataire N_D. Le noeud destinataire N_D peut initialement contenir ou non un *dispatcher*.

5.4.2 *Eclatement de dispatcher*

L'Eclatement de *dispatcher* est une action de reconfiguration que nous modélisons par une règle de réécriture des graphes. Cette règle est représentée comme le montre la figure 6 :

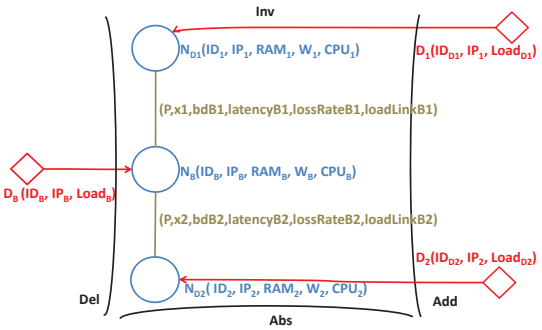

Figure 6: Règle de l'Eclatement d'un *dispatcher*

Cette règle (figure 6) consiste à éclater un *dispatcher* D_B déployé sur un noeud dégradé N_B en deux *dispatchers* D_1 et D_2 vers deux noeuds destinataires N_{D1} et N_{D2}.

5.4.3 *Substitution d'un lien logique*

La Substitution d'un lien logique est une action de reconfiguration que nous modélisons par une règle de réécriture des graphes. Cette règle est représentée le montre la figure 7 :

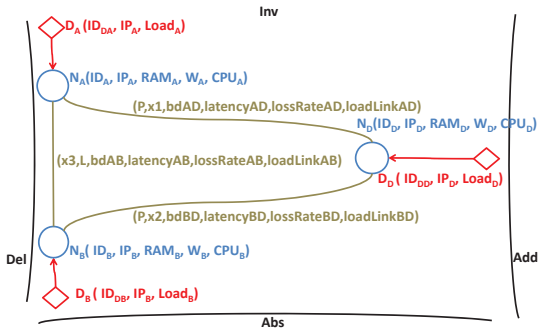

Figure 7: RG de la substitution d'un lien logique

Cette règle (figure 7) consiste à substituer un lien logique entre D_A et D_B. Ces derniers sont déployés sur les noeuds N_A et N_B. Le *dispatcher* intermédiaire à intercaler est D_D déployé sur le noeud N_D.

6. IMPLANTATION

Les étapes de travail consistent à intervenir aux dernières étapes du processus de reconfiguration comme est déjà expliqué dans la sous-section 5.3. Nous presentons les configurations retenues qui résultent de l'application des politiques de reconfiguration sur le graphe G_0 mentionné dans la section 4. Notre approche permet de retrouver le graphe G_s issue des actions de reconfigurations a partir de G_0 ($G_0 \xrightarrow{Reconfigurations} G_s$).

Supposant que le graphe initial (figure 4) est représenté par : $G_0(1*1,0)/(1*1,0),(1*2,0),(1*3,0)$ et que les dégradations sont détectés au niveau du noeud dégradé N_1 et du lien dégradé (D_2-D_4) souffrant d'un problème de charge (figure 4). L'application des politiques de reconfiguration reviennent à l'application de différents algorithmes de traitement des problèmes détectés. Les Problème_Noeuds puis les Problème_Liens sont traités en appliquant toutes les combinaisons des actions de reconfiguration. Ensuite des critères d'évaluation sout appliqués sur les graphes et un graphe de sortie est sélectionné.

Les résultants de l'application des politiques de reconfiguration au graphe initial donne un arbre de reconfiguration. En appliquant la selection sur les graphes des états finaux de cet arbre nous obtenons les graphes de sortie suivant :
G^1_{A1} := G^1_1 de l'arbre A_1,((0*1,1)/((1*1,0),(0*2,0),(0*3,0)))
G^2_{A1} := G^2_1 de l'arbre A_1,((0*1,1)/((1*1,0),(0*2,0),(0*3,0)))
G^1_{A2} := G^1_1 de l'arbre A_2,((0*1,1)/((0*1,1),(0*2,0),(0*3,0)))
G^2_{A2} := G^2_1 de l'arbre A_2,((0*1,1)/((0*1,1),(0*2,0),(0*3,0)))

Il est possible de déduire que l'application des différentes politiques au graphe initial mènent à des graphes de sorties présentant différents états. Donc nous appliquons nos critères d'évaluation pour choisir une sortie. L'application de nos critères d'évaluation sur ces graphes d'états finaux nous mènent à avoir en sortie G^1_{A2} et G^2_{A2} car leurs problèmes sont totalement réparés.

7. CONCLUSION

Dans cet article, nous avons détaillé les problèmes qui peuvent survenir dans un système Publier/Souscrire déployé sur un réseau MANET. Face à ces problèmes, nous avons proposé un processus de reconfiguration dynamique mettant en place différentes politiques d'adaptation dans le but d'explorer l'espace des solutions d'adaptation possibles. Nous avons modèlisés notre système en utilisant les graphes pour la partie MANET (Physique) et pour la partie Publier/-Souscrire (Logique). Nous avons utilisés les règles de transformation de graphes pour modèliser les actions de reconfigurations. L'évaluation de ces actions de reconfiguration sera utile pour la phase de planification qui à chaque cause d'échec du système, fait correspondre l'action de reconfiguration adéquate. Nous comptons dans les travaux futurs, évaluer ces politiques d'adaptation en mode hors-ligne, pour pouvoir définir des politiques plus efficaces

8. REFERENCES

[1] M. Avvenuti, A. Vecchio, and G. Turi. A cross-layer approach for publish/subscribe in mobile ad hoc networks. In *Mobility Aware Technologies and Applications, Second International Workshop, MATA 2005, Montreal, Canada, October 17-19, 2005, Proceedings*, pages 203–214, 2005.

[2] I. Bouassida Rodriguez, R. Ben Halima, K. Drira, C. Chassot, and M. Jmaiel. A graph grammar-based dynamic reconfiguration for virtualized web service-based composite architectures. In *Business System Management and Engineering*, pages 181–196. Springer Berlin/Heidelberg, 2012.

[3] N. Carvalho, F. Araujo, and L. Rodrigues. Scalable qos-based event routing in publish-subscribe systems. In *Fourth IEEE International Symposium on Network Computing and Applications (NCA 2005), 27-29 July 2005, Cambridge, MA, USA*, pages 101–108. IEEE Computer Society, 2005.

[4] M. K. Denko. Pusman : Publish-subscribe middleware for ad hoc networks. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering, CCECE 2007, May 7, 10, 2006, Ottawa Congress Centre, Ottawa, Canada*, pages 1677–1681. IEEE, 2006.

[5] G. P. P. Gianpaolo Cugola. Reds : a reconfigurable dispatching system. In *Proceedings of the 6th international workshop on Software engineering and middleware*, pages 9–16, 2006.

[6] V. Gnanasekaran. Evaluating application architecture, quantitatively. *The Architecture Journal*, 2010.

[7] K. Minkyong, K. Kyriakos, R. Johnathan, L. Hui, and S. Konstantin. Efficacy of techniques for responsiveness in a wide-area publish/subscribe system. In *Proceedings of the 11th International Middleware Conference Industrial track*, pages 40–45, 2010.