# Application Driven Networking in Dynamic Environments - Cas d'application simulation distribuée : implémentation, validation et évaluation

Slim Abdellatif, Armel Francklin Simo Tegueu, Mikaël Capelle, Pascal Berthou, Thierry Villemur, Marie-José Huguet

Projet ANR- 13-ASTR-0024 ADN

*Projet ANR- 13-ASTR-0024*

Application Driven Networking in Dynamic Environments

## Livrable 1.2

# Architecture préliminaire et algorithmes pour le cas d'applications dynamiques en environnement maîtrisé

| | |
|---|---|
| Référence document | ANR-13-ASTR-0024-L1.2 |
| Liste des auteurs | S. Abdellatif, F. Simo Tegue, P. Berthou, T. Villemur<br><br>[a] CNRS, LAAS, 7 avenue du colonel Roche, F-31400<br>[b] Univ de Toulouse, INSA, LAAS, F-31400, Toulouse, France<br>[c] Univ de Toulouse, LAAS, F-31400, Toulouse, France<br>[d] Univ de Toulouse, LAAS, F-31400, Toulouse, France<br>[e] Univ de Toulouse, UPS, LAAS, F-31400, Toulouse, France<br>[f] Univ de Toulouse, UTM, LAAS, F-31100, Toulouse, France |
| Version | finale |
| Date de livraison | Juillet 2015 |

# Table des matières

# Introduction

L'objectif général du projet ADN (Application Driven Networking) est de contribuer à développer le concept de réseau guidé [1] (noté ci-après par *réseau ADN*). En d'autres termes, développer un réseau capable de fournir *à chaque* application un service d'acheminement personnalisé qui répond et colle à ses besoins en provisionnant des ressources instantanément et automatiquement selon les besoins de l'application. Les applications visées par ce type de réseau sont dynamiques et exigent une certaine qualité de service (QoS pour Quality of Service) qui peut être plus ou moins stricte. Le caractère dynamique de ces applications signifie que les flux applicatifs échangés entre les processus de l'application peuvent évoluer dans le temps et que les exigences de QoS (en termes de débit, délai, etc.) pour délivrer ces flux peuvent, à leur tour, changer dans le temps.

Pour offrir un service d'acheminement personnalisé, un réseau ADN repose sur un acheminement basé flux (et non pas basé sur une adresse identifiant un terminal destinataire ou un groupe de terminaux). En d'autres termes, à la réception d'un paquet, un noeud d'un réseau ADN :

- identifie l'application et/ou le flux applicatif auquel appartient le paquet (et non pas le ou les noeuds terminaux destinataires) ;
- déduit les ressources réseau qui lui sont destinées ou qui lui sont nécessaires ;
- Mobilise ces ressources pour que le paquet reçoive le traitement attendu.

Plusieurs propositions de réseau ADN émergent dans la littérature scientifique. Elles se distinguent par les approches adoptées pour inférer, d'une part, l'application à laquelle appartient le paquet et, d'autre part, les besoins de QoS à satisfaire et les ressources correspondantes à mobiliser. Dans le cadre du projet ADN qui vise le support d'applications DDS (Data Distribution Service) dynamiques avec des exigences plutôt strictes de QoS, l'approche adoptée repose sur le choix suivant : les besoins d'une application sont explicitement communiquées au (ou capturées par le) réseau (via le middleware DDS) avec un niveau de granularité très fin. Une infrastructure réseau de type SDN(Software Defined Network) est supposée afin de programmer à la volée le réseau de sorte à offrir dynamiquement le service réseau souhaité par l'application.

L'objectif de se rapport est de présenter l'architecture préliminaire de la solution réseau ADN développée dans le cadre de ce projet pour le cas d'une infrastructure réseau SDN filaire. Les algorithmes des principaux composants de l'architecture sont également décrits.
Le plan de ce rapport se présente comme suite. Le chapitre 1 présente l'architecture préliminaire du réseau ADN développé dans ce projet ainsi qu'une analyse fonctionnelle des principaux composants de l'architecture. Le chapitre 2 décrit l'algorithme de capture des besoins applicatifs et leur retraduction en une demande de service réseau. Le chapitre

1. également appelé réseau centré ou conscient des applications

3 décrit l'algorithme (dit d'allocation de ressources) qui calcule les ressources réseaux nécessaires pour offrir les services réseaux attendus. Le chapitre 4 présente l'algorithme (dit de déploiement des services) qui provisionne les ressources calculées sur les éléments de l'infrastructure SDN. Ce rapport s'achève par une conclusion qui dresse les spécificités et les points forts de l'approche développée dans le cadre du projet ainsi que les pistes identifiées pour la suite de travail.

# Chapitre 1

# Architecture Préliminaire

## 1.1 Architecture générale du réseau ADN

Le réseau ADN développé dans le cadre de ce projet a pour objet d'offrir des services réseau personnalisés pour des applications bâties sur un middleware DDS. Avec DDS, il est possible de caractériser *très finement* les besoins d'une application : les flux applicatifs et leurs besoins en QoS. En effet, reposant sur un modèle de coopération en "publish/subscribe", une application DDS est une collection de processus (ou programmes) applicatifs qui produisent des données auxquelles d'autres processus applicatifs souscrivent pour les consommer. Ainsi, à partir de la liste des producteurs des différentes données applicatives et de leurs consommateurs respectifs, il est possible de connaître à tout moment les flux de données échangés avec leur source et leur(s) destination(s). Puisque DDS permet aux applications de spécifier les exigences de QoS relatives à la distribution des différentes données applicatives [17], il est également possible de déterminer, à tout moment, les besoins en QoS associés aux flux de données (exprimées en termes de débit et délai).

La ligne directrice de notre approche est d'offrir un service qui colle au plus près aux besoins des applications, c'est-à-dire, satisfaire leurs exigences tout en utilisant au mieux les ressources réseau. De ce fait, l'idée est de raisonner à l'échelle des flux de données suscitées pour déduire les caractéristiques du service réseau (avec les ressources réseau nécessaires) à provisionner. Ce service correspond à un réseau virtuel overlay [1] constitué d'un ensemble de liens virtuels de bout-en-bout (entre producteurs et consommateurs de données) qui peuvent être de type point-à-point ou point-multi-point et sont caractérisés par un débit et un délai de transfert maximal. Etant donné que les flux de données (et leurs besoins) sont susceptibles de varier dans le temps, le réseau overlay à provisionner est à son tour dynamique aussi bien au niveau de sa topologie que la capacité de ses liens. Nous reposons sur une infrastructure réseau de type SDN qui, par sa possibilité d'être programmée en temps-réel, permet d'envisager le support efficace de services dynamiques.

La figure 1.1 résume l'architecture générale ADN. Cette dernière met en évidence l'application de contrôle réseau (ou fonction réseau) *"Provisionnement et maintien de réseaux virtuels pour applications DDS "* dont le rôle est de provisionner des services overlay pour des applications DDS sur une infrastructure de type SDN/Openflow. Cette fonction s'interface d'un côté avec le middleware DDS afin de capturer les besoins ap-

1. nous utiliserons dans ce qui suite le terme réseau virtuel

plicatifs et, de l'autre côté, avec le contrôleur SDN pour déployer les service associés.

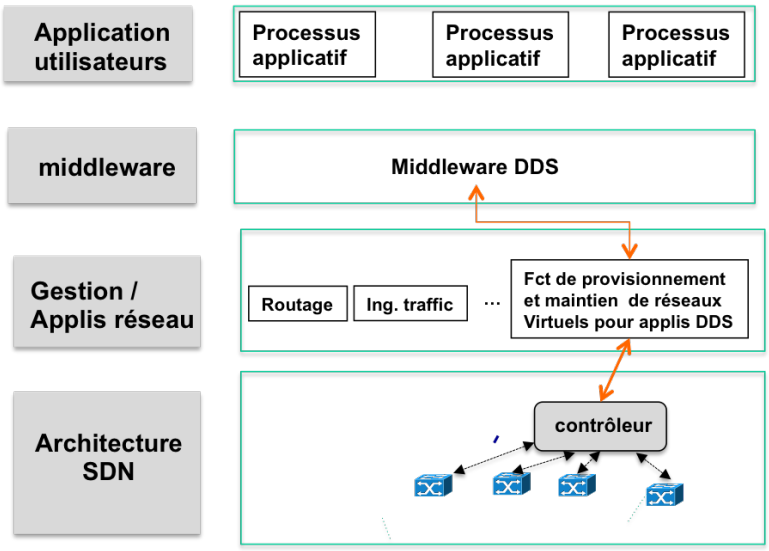


FIGURE 1.1 – Architecture générale

La figure 1.2 présente de manière générale les différentes alternatives qui peuvent être envisagées pour développer la fonction de provisionnement. Une première possibilité serait de reposer sur une interface nord du contrôleur de haut niveau : Soit, en utilisant des langages de programmation réseau de haut-niveau tels que par exemple pyretic [22] ou Procera [25], soit, en reposant sur des services réseau offerts par certains contrôleurs SDN (certains contrôleurs offrent des services capables de provisionner des réseaux virtuels sur une infrastructure SDN).

Ces précédentes alternatives présentent néanmoins des limites par rapport à nos attentes et notamment, l'impossibilité de provisionner des liens virtuels point-multipoints et de spécifier des exigences qui combinent le délai et le débit. De plus, en l'état actuel des choses, ils reposent sur des algorithmes d'allocation de ressources basiques (qui seront certainement amenés à évoluer) et dont certains n'ont pas comme objectif principal l'utilisation optimale des ressources réseau. En conséquence, le choix effectué dans ce travail est de reposer sur une interface nord de bas niveau où l'on spécifie au contrôleur les règles Openflow à installer au niveau de chaque commutateur de l'infrastructure SDN (ce choix est représenté sur la figure 1.2 par la flèche pleine orange). De ce fait, nous faisons le choix de ne pas utiliser un hyperviseur réseau qui aurait pu prendre en charge le déploiement du réseau virtuel et la ségrégation entre réseaux virtuels. Dans ce travail, ce déploiement est réalisé par les règles Openflow produites par notre fonction de provisionnement ; La ségrégation est assurée par la cohérence entre les règles Openflow générées pour des réseaux virtuels différents. Une analyse de la pertinence d'utiliser un hyperviseur réseau mérite d'être considérée dans la suite des travaux. Elle se traduirait par l'analyse des possibilités des principaux hyperviseurs du marché et une confrontation par rapport à nos besoins.

Sur la base des choix décrits ci-avant, la fonction de provisionnement se compose des éléments suivants (illustrés dans la figure 1.3)[20, 19] :

— **Capture des besoins applicatifs et re-traduction en service réseau** dont

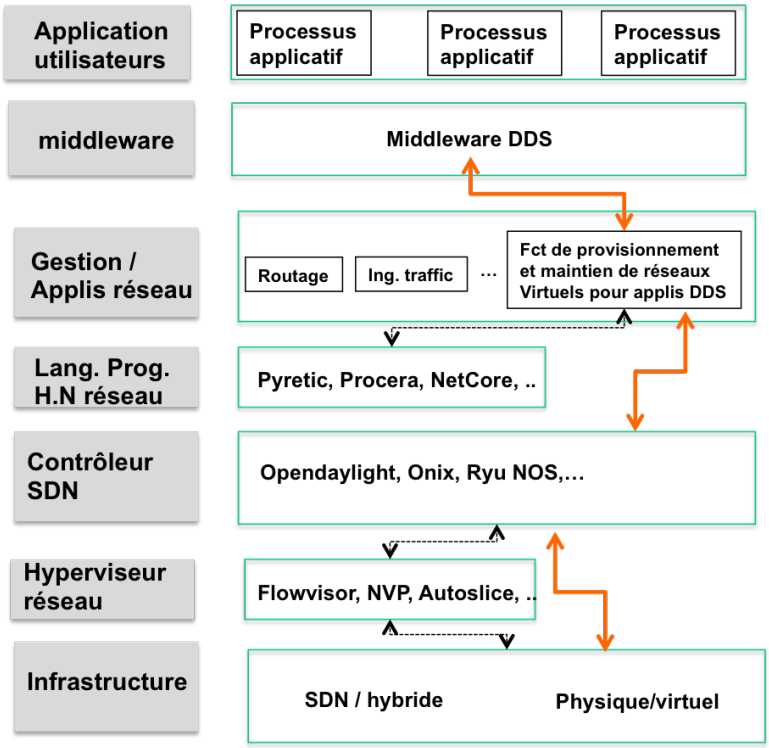


FIGURE 1.2 – Choix de conception relatifs à la fonction de provisionnement

l'objectif est, d'une part, collecter les besoins d'une application DDS et, d'autre part, de les retraduire en une demande de réseau virtuel avec les caractéristiques appropriées (topologie, débit, délai, etc.). Cet élément peut être une partie intégrante du middleware DDS ou bien être externe à celui-ci. Le premier cas de figure est le plus naturel puisque le middleware DDS a la connaissance complète des besoins des applications (puisqu'elles le spécifient explicitement à travers l'API DCPS du middleware DDS) et peut donc exprimer directement au réseau les services qu'il souhaite. En revanche, sa mise en oeuvre nécessite d'intervenir directement sur le code des middlewares DDS.
Dans le deuxième cas de figure, cet élément joue le rôle d'une application DDS de monitoring d'un domaine DDS dont le but est de capturer les événements DDS relatifs aux demandes de souscription/dé-souscription à des topics, des demandes de changement QoS, etc. La mise en oeuvre de ce composant est simplifié (non intrusif sur les middleware DDS). Il est transparent pour le middleware (et est capable de fonctionner avec différentes implémentations du middleware DDS), mais a un coût en termes de sur-débit. C'est ce dernier cas de figure que nous adoptons pour la suite.

— **allocation de ressource pour RV (réseau virtuel)** qui à partir de la demande de réseau virtuel ci-avant, a pour tâche de calculer les chemins physiques et les ressources (équipements et liens traversés) supports du réseau virtuel
— **déployeur du RV** qui à partir des ressources déterminées par l'*allocateur de ressources* a pour mission de prendre en charge le déploiement effectif du réseau virtuel sur l'infrastructure réseau en générant les règles Openflow qui seront appliquées aux commutateurs de l'infrastructure
— **Monitor réseau** qui pour le cas d'une infrastructure réseau filaire a pour objectif de détecter des changements de topologie de l'infrastructure réseau et de les

notifier au module de gestion décrit ci-après. Ces changements sont typiquement provoqués par des défaillances de certains éléments du réseau.

— **gestion autonomique** dont l'objectif est de rendre la fonction de provisionnement autonomique. En analysant les informations remontées par le *Monitor réseau*, il analyse la situation et en cas de besoin planifie et exécute les actions correctrices appropriées en agissant sur le module d'*"allocation de ressources"* ou le module *capture des besoins applicatifs et re-traduction en service réseau.*

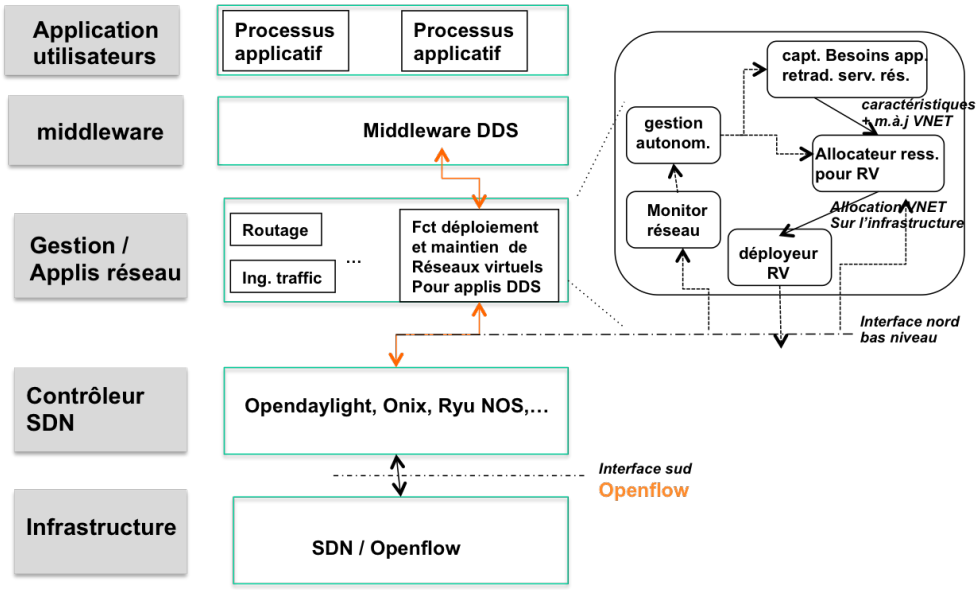


FIGURE 1.3 – Composants de la fonction de provisionnement

Il est à noter que les différents éléments décrits ci-avant sont des composants logiciels à part entière qui peuvent être localisées sur des machines différentes. Certains peuvent même être des composants pris sur étagère : cela pourrait par exemple être le cas du composant *Monitor réseau*.

Dans la suite, nous présentons une description un peu plus détaillée des principales fonctions qu'implémentent les composants "*Composant de capture des besoins applicatifs et de re-traduction en service réseau*", "*Allocateur de ressources pour réseau virtuel*", "*déployeur de réseau virtuel*" et "*Monitor réseau*". Les algorithmes qu'exécutent ces fonctions sont décrits dans les chapitres suivants. L'analyse fonctionnelle du composant "*Gestion autonomique*" n'est pas présentée. A la date d'écriture de ce rapport, nous ne disposons pas d'une liste précise des exigences fonctionnelles de ce composant. Ce travail est l'objet de nos actions en cours et sera finalisé pour le livrable 1.3 décrivant l'architecture finale de ADN.

## 1.2 Composant de capture des besoins applicatifs et de re-traduction en service réseau

L'objectif de ce module est de surveiller le domaine DDS afin de capturer les événements DDS qui ont un impact sur les flux de données échangés et leurs besoins (les demandes de création/destruction/souscription/dé-souscription à des topics, des demandes de changement QoS, etc.) pour ensuite les retraduire en une demande de création, de modification ou de suppression d'un réseau virtuel applicatif.

Pour ce faire, ce composant exploite le service de monitoring fourni aux applications par DDS à travers les **Built-In Topics** et les interfaces Listeners associées aux différentes entités du domaine DDS (topic, producteur, souscripteur, *Data Writer*, *Data Writer*)[18]. Ce service de monitoring permet de notifier automatiquement les applications l'occurrence d'évènements touchant les entités DDS mis sous surveillance par l'application et d'exécuter des procédures (i.e. méthodes) de *callback* prévues à cet effet. A titre d'exemple, en activant le monitoring sur les opérations de souscription, le composant *capture des besoins applicatifs* est notifié de chaque demande de souscription réussie à un topic, il peut donc enclencher le calcul de la nouvelle topologie du réseau virtuel applicatif.

Comme expliqué ci-avant, l'approche que nous adoptons pour la capture des besoins applicatifs est transparente pour l'application et le middleware, renforçant l'adoption et l'intégration de notre architecture dans les systèmes existants. Le prix à payer est le sur-débit occasionné : le bloc fonctionnel du composant exécute le middleware DDS et participe activement au domaine DDS en tant qu'observateur (donc consommateur) de certains événements DDS.

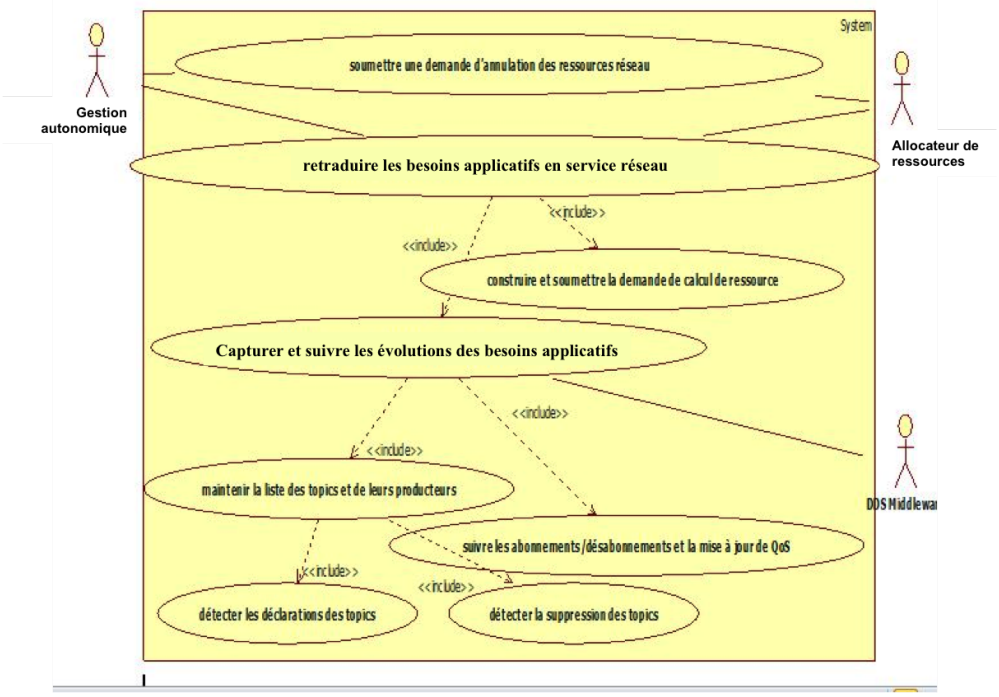


FIGURE 1.4 – Diagramme des cas d'utilisation du composant "capture de besoins et retraduction en service réseau"

La figure 1.4 décrit le diagramme des cas d'utilisation du composant qui résume ses principales fonctionnalités discutées ci-avant. La fonction *construire et soumettre la demande de calcul de ressources* a, entre autres, pour mission de retraduire les besoins applicatifs en une demande de réseau virtuel constitué d'un ensemble de liens virtuels de bout-en-bout. Il est clair que cette opération n'est pas triviale car notre souhait de reposer sur une connaissance très fine des besoins des applications (en raisonnant à l'échelle de flux de données élémentaires) peut avoir un impact sur la complexité de la topologie du réseau virtuel et donc sur la résistance à l'échelle (*"scalabilité"*) de notre approche. Cette fonction repose donc sur un problème d'optimisation combinatoire dont le but est de définir les meilleurs regroupements de flux afin de réduire le nombre de liens

virtuels qui constituent une demande de réseau virtuel. La formulation de ce problème d'optimisation est l'objet de nos prochaines actions et sera décrite dans le livrable de l'architecture finale ADN.

## 1.3 Allocateur de ressources pour réseau virtuel

L'objectif principal de ce composant est de calculer à la volée les chemins physiques (avec les ressources réseau à allouer le long de ces chemins) support de chaque réseau virtuel applicatif. Chaque lien est caractérisé par une bande passante et un délai maximal à garantir et peut être de type point-à-point ou point-multipoints. L' *allocateur de ressources* considère deux types de ressources réseau : classiquement, les ressources de transmission (capacité des liens de transmission) et les ressources de commutation au niveau des commutateurs SDN traversés. La prise en compte de ce dernier type de ressources est primordiale puisqu'avec les technologies actuelles, l'opération de commutation dans un réseau SDN est consommatrice en temps ce qui impacte la taille maximale des tables d'acheminement des commutateurs SDN. L'algorithme de l'*allocateur* cherche à minimiser la quantité de ressources (transmission et commutation) allouées à chaque réseau virtuel ainsi qu'à équilibrer la charge entre les différents nœuds et liens de l'infrastructure physique. De ce fait, il favorise l'admissibilité des prochaines demandes d'allocation de ressources. Au besoin, il offre la possibilité d'activer une fonction de « path-splitting » qui ouvre le champ à des allocations de ressources d'un lien virtuel sur plusieurs chemins physiques. Afin d'éviter la dispersion des allocations sur une multitude de chemins physiques, le « path-splitting » n'est permis que si les allocations demeurent supérieures à un seuil minimal prédéfini.

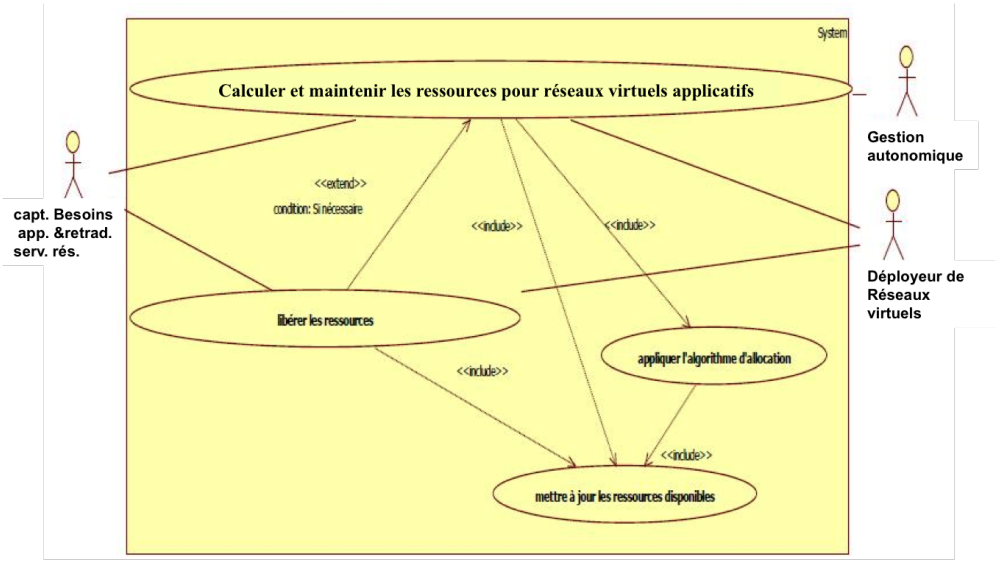


FIGURE 1.5 – Diagramme des cas d'utilisation du composant allocateur de ressources

La figure 1.5 présente les principales fonctionnalités de ce composant qui met en exergue les différentes situations où cette fonction est invoquée : au lancement de l'application, suite à un changement des besoins applicatifs (flux de données ou QoS), à la demande du module de gestion autonomique, en réponse à un symptôme décelé.

## 1.4 Monitor réseau

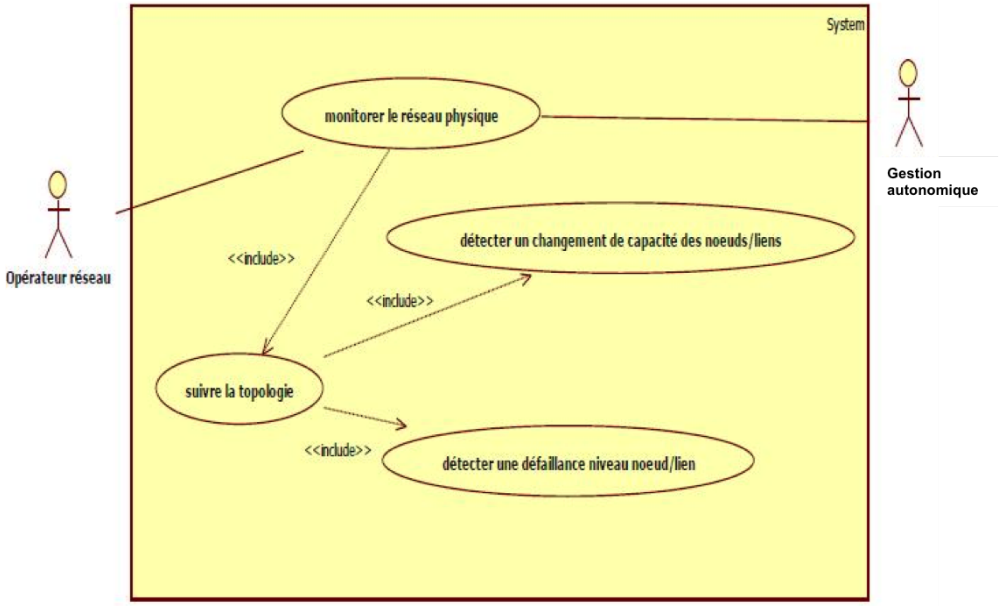


FIGURE 1.6 – Diagramme des cas d'utilisation du composant Monitor réseau

L'objectif de ce composant est de suivre l'évolution de la topologie physique du réseau, des caractéristiques des liens physiques (capacité, charge et délai) et des nœuds (capacité, charge), et de surveiller en temps-réel la disponibilité des ressources réservées à chaque réseau virtuel applicatif pour remonter toute anomalie au module de gestion autonomique.

L'OpenFlow Notification Framework [4], énumère une liste d'évènements, en plus de ceux déjà évoqués dans les spécifications OF-Config [5] et OpenFlow Switch tels que : link failure, configuration change, port status, Flow removed etc... notifiés au contrôleur et exploités par ce module via son interface Nord, pour suivre l'évolution de la topologie et la charge des nœuds (nombre d'entrées actives dans les tables).

La surveillance de la bande passante réservée à un lien virtuel, se fait en utilisant la technique de monitoring en mode actif, consistant à interroger périodiquement les statistiques des entrées actives des tables associées à ces liens, grâce au concept de compteur et des messages tels que : queue statistics message et meter statistics message permettant leur interrogation à travers l'API Nord du contrôleur SDN.

## 1.5 Déployeur de réseau virtuel

L'objectif de ce composant est d'installer sur les commutateurs de l'infrastructure SDN/Opeflow les règles OpenFlow qui permettent de déployer ou de mettre à jour un réseau virtuel applicatif. Cela se traduit par une liste de blocs d'instructions rédigées conformément à l'API Nord du contrôleur, à livrer par ce dernier à chaque commutateur concerné via le protocole OpenFlow. Ce module s'appuie sur la propriété transactionnelle du réseau fournie par la spécification OpenFlow 1.4.0 via l'introduction des bundles. Cette propriété permet de contrôler le caractère atomique de ces instructions, afin de garantir une cohérence entre elles et l'état du réseau, évitant également les pertes

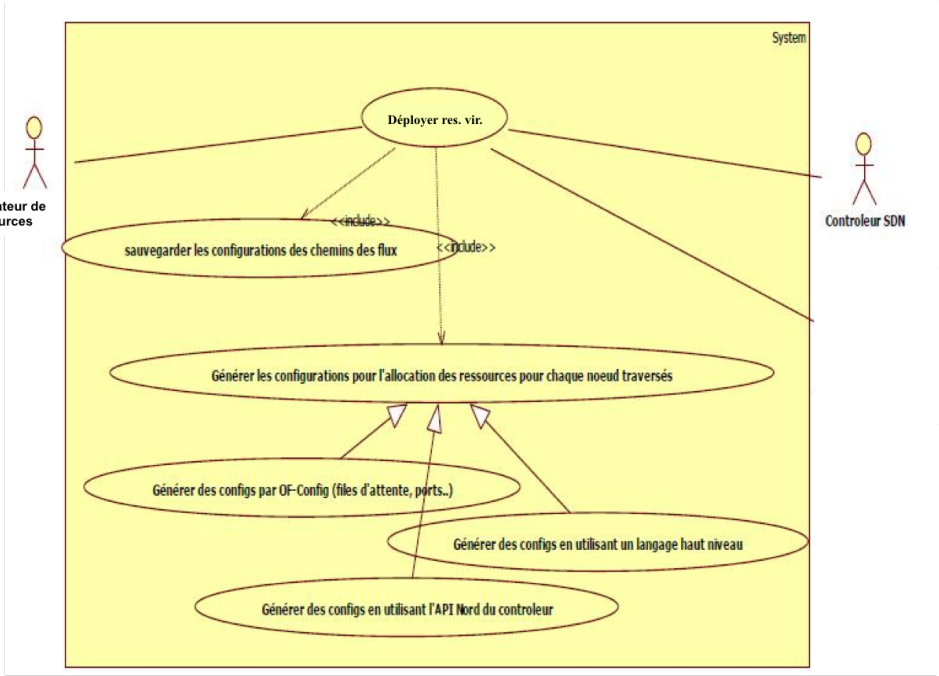


FIGURE 1.7 – Cas d'utilisation du composant Déployeur réseau virtuel

éventuelles de ressources, et ce, sans interruption de transmission pour une plus grande fiabilité et continuité de service.
L'identification d'un lien virtuel repose classiquement sur les champs adresses IP et numéros de port des paquets. En effet, ce sont ces identificateurs numériques qui sont utilisés par DDS pour identifier les *Data Writer* et *Data Reader*. Ainsi, les règles de correspondances (matching) des règles Openflow reposeront sur une combinaison de ces champs. Les débits alloués aux liens virtuels seront régulés par des meters Openflow.

# Chapitre 2

# Capture des besoins applicatifs et retraduction en une requête de service réseau

## 2.1 Introduction

Nous présentons successivement les algorithmes des deux fonctions du composant : la capture des besoins applicatifs et la re-traduction de ces besoins en une requête de service réseau.

## 2.2 Capture des besoins applicatifs

L'algorithme de capture des besoins applicatifs repose sur le service de découverte DDS dont le but est de permettre à tout processus applicatif participant à un domaine DDS de déclarer, d'une part, la production d'un ou plusieurs flux de données (i.e. *Topic DDS*) (avec une certaine QoS), et d'autre part, son souhait de souscription à des flux de données (avec une QoS requise). Le middleware DDS se charge alors de mettre en correspondance automatiquement les souscripteurs et producteurs d'un même topic (sous l'hypothèse que la QoS côté producteur soit en phase avec la QoS requise côté souscripteur). C'est en capturant ces annonces de production, souscription, désouscription, etc., que notre algorithme est capable de construire une vue précise des flux de données échangés et de leurs exigences de QOS.
Dans ce qui suit, nous décrivons le service de découverte de DDS pour en déduire la méthode utilisée pour capturer les besoins applicatifs.

### 2.2.1 Le service de découverte de DDS

Avec le middleware DDS, la mise en correspondance entre un producteur d'un flux de données (i.e. Topic) avec son (ses) consommateur(s) se base conjointement sur (1) le nom de topic, (2) son type et (3) sur la vérification que la QoS offerte [1] par le producteur répond (i.e. plus stricte) à la QoS requise par le souscripteur [2]. Elle est automatique et

1. se référer à [17, 23] pour la liste des paramètres de QoS
2. en effet, la QoS côté producteur peut être différente de celle des souscripteurs

peut intervenir à tout moment et pas nécessairement au démarrage/à l'arrêt de l'application ou du producteur, offrant ainsi un découplage dans le temps entre un producteur de topic et les souscripteurs associés. DDS intègre un service de découverte qui permet, à tout moment, aux producteurs de topics de se déclarer sur le réseau et d'y annoncer les topics (et la QoS offerte) qu'ils produisent et similairement aux souscripteurs de se déclarer sur le réseau et d'y annoncer les *topics* (avec la QoS requise) auxquels ils souhaitent souscrire. Ce service permet également d'annoncer sur le réseau tout autre changement dont une désouscription, arrêt de production ou un changement de QoS.
DDS n'impose aucun protocole pour implémenter ce service, ouvrant le champ à une multitude de solutions (selon le contexte (par exemple : pour LAN ou pour WAN, pour du filaire ou du sans-fil, etc.) qui pouvent, potentiellement, être en action en même temps. Le standard DDS impose néanmoins l'intégration à tout middleware DDS d'une version simple du protocole de découverte, à savoir le protocole **Simple Discovery Protocol (SDP)**. En pratique et par défaut, c'est le seul protocole que l'on retrouve dans les middlewares DDS.
Nous rappelons dans ce qui suit quelques éléments de terminologie DDS avant de décrire le principe de fonctionnement des protocoles de découverte (tq décrits par le standard DDS [17]) et de terminer la section par la présentation du protocole SDP (que suppose et utilise notre algorithme de capture des besoins).

#### 2.2.1.1 Quelques éléments de terminologie DDS

Nous rappelons ci-après les éléments architecturaux clefs de la spécification DDS (se référer à la figure 2.1) :

- "Domaine DDS" : est un concept qui permet de confiner la distribution de données au sein d'un groupe d'applications qui nécessairement se partagent les mêmes intérêts (topics). Un domaine DDS est défini par un *domain ID* ;
- "Participant d'un domaine" (*Domain participant*) : matérialise l'appartenance d'une application à un domaine DDS et englobe des entités de domaine DDS *domain entities*, typiquement des producteurs (*publier*) et/ou souscripteurs (*subscriber*) ;
- "entités DDS terminales" (*endpoint entities*) : ce sont les *Data Writers* et *Data Readers* qui, respectivement, écrivent et lisent les données depuis l'espace de données partagé pour le compte des applications. Ils sont respectivement rattachés à un producteur et un souscripteur du *domaine participant* de l'application.

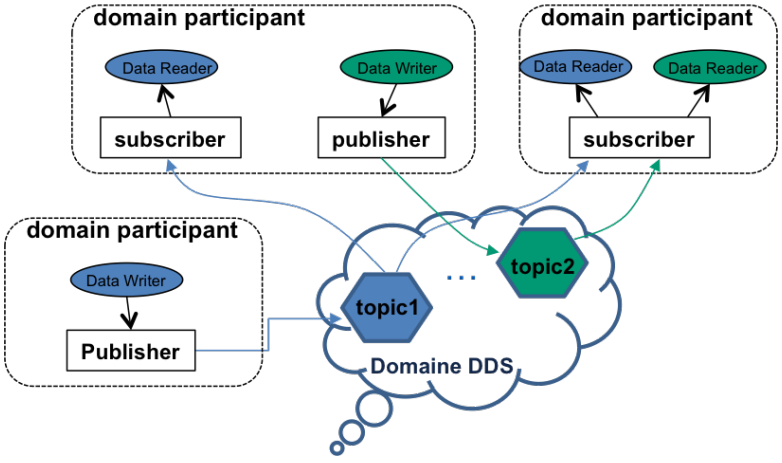


FIGURE 2.1 – Architecture DDS

#### 2.2.1.2 Principe général des protocoles de découverte DDS

Le principe général des protocoles de découverte est défini dans le standard de l'OMG relatif au protocole *Real-Time Publish Subscribe* (RTPS) [18]. Ce standard découpe le protocole de découverte en deux protocoles indépendants, chacun en charge d'une phase précise du processus de découverte :

- **Participant** Discovery Protocol (PDP) : ce protocole permet la découverte mutuelle des participants (au sens *domain participant*) à un domaine DDS qui sont connectés au réseau ;
- **Endpoint** Discovery Protocol (EDP) : une fois la découverte de participants terminée, le protocole EDP rentre en jeu pour découvrir les entités terminales (*endpoints*) qui y sont rattachées. Cela permet à chaque participant DDS de se constituer une base d'information relative à chaque *endpoint* (topics, QoS, localisation sur le réseau, etc.).

Sur la base des informations collectées sur les *endpoints* (qui est la finalité du service de découverte), la phase de mise en correspondance entre *Data Writers* et *Data Readers* compatibles peut intervenir avec, au final, la configuration du canal de communication support des échanges de données. Cette phase ne concerne pas le service de découverte et est propre à chaque implémentation DDS.

Les protocoles de découverte DDS utilisent des mécanismes DDS pour supporter les échanges de leurs messages d'annonce ; Ils n'ont donc pas recours à (ou ne supposent pas l'existence) des mécanismes de communication hors-bande. Plus précisément, ils reposent sur l'utilisation (voir figure 2.2) :

- d'entités DDS terminales intégrées par défaut à chaque participant à un domaine DDS lors de sa création, appelés *built-in Discovery endpoints* (ou plus simplement *built-in entities*). Ces *built-in Data Readers* et *built-in Data Writers* ont respectivement pour charge d'annoncer les informations relatives à leur *domain participant* d'appartenance et de collecter les annonces émanant des autres *domain participants* présents sur le réseau ;
- de quatre topics DDS intégrés par défaut à tout domaine DDS à son initialisation, appelés les *built-in topics* : **DCPSParticipant**, **DCPSSubscription**, **DCPSPublication** et **DCPSTopic**. Ces *topics* représentent les informations échangées dans les annonces citées ci-avant. Ainsi, pour chacun, il existe potentiellement, au niveau de chaque participant DDS présent sur le réseau, un *built-in Data Writer* et un *built-in Data Reader* (voir figure 2.2).

#### 2.2.1.3 Le protocole SDP

SDP reprend évidemment l'un des principes des protocoles de découverte DDS, à savoir la décomposition de celle-ci en deux étapes successives, chacune assurée par un protocole spécifique. Les protocoles de découvertes SDP sont donc : le **Simple Participant Discovery Protocol (SPDP)** et le **Simple Endpoint Discovery Protocol (SEDP)**. Nous les décrivons ci-après.

**2.2.1.3.1 Simple Participant Discovery Protocol** Le protocole SPDP suit une approche assez simple pour annoncer et déceler la présence sur le réseau de participants à un domaine DDS, à savoir : la distribution périodique selon le modèle *best-effort* des données de type *DCPSParticipant* à une liste pré-configurée d'adresses de machines (*locators list*). Cette liste est le moyen de localiser, à tout moment, les éventuels participants

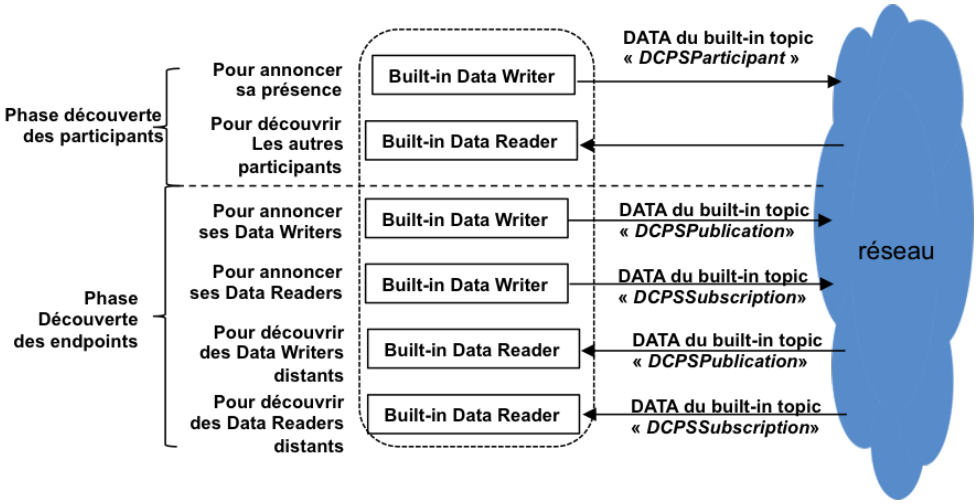


FIGURE 2.2 – Protocole de découverte DDS

présents sur le réseau. Elle peut inclure aussi bien des adresses unicast que multicast. L'envoi périodique des données relatives à chaque participant donne l'occasion à un participant qui rejoint le réseau de prendre connaissance des participants déjà rattachés au domaine. Par ce biais (au prix d'un sur-débit accru), SPDP n'impose aucun ordre ni aucune contrainte sur les arrivées de chaque participant.

Les données SPDP échangées entre les participants sont définies par le topic *SPDPdiscoveredParticipantData* qui étend le built-in topic *DCPSParticipant*, évoqué ci-avant. Les principaux attributs sont décrits dans le tableau 2.1.

Similairement, le *Data Writer* et *Data Reader* (intégrés par défaut dans tout participant) qui permettent respectivement d'écrire et de lire des données *SPDPdiscoveredParticipantData* sont une spécialisation des *built-in entities* génériques cités dans la section 2.2.1.2. Il s'agit de : *SPDPbuiltinParticipantWriter* et *SPDPbuiltinParticipantReader*. Parmi les caractéristiques de ces entités terminales, on retrouve, pour chacun, la liste des adresses de localisation unicasts et multicasts utilisés pour alimenter les messages échangées par les participants. Ces adresses sont configurées automatiquement par le middleware DDS ou configurées automatiquement par l'application.

**2.2.1.3.2 Simple Endpoint Discovery Protocol** L'objectif du protocole SEDP est de permettre la découverte des *Data Writers* et *Data Readers* (et par conséquence : les topics auxquels ils s'intéressent, la QoS associée et leur localisation) présents au niveau des différents participants d'un domaine. SEDP définit le format des messages échangés et les entités terminales (dans les participants) impliqués dans ces échanges. Comme pour le protocole SPDP, SEDP repose sur la distribution de données DDS pour réaliser ces échanges (avec une petite nuance : une distribution *fiable* des messages). Ainsi, comme préconisé dans la section 2.2.1.2, SEDP définit un ensemble de *built-in topics* et des *built-in endpoints* qui produisent et consomment les données de ces topics. L'ensemble est décrit ci-après.

SEDP définit trois types de données intégrés à tout middleware DDS et instanciés pour chaque domaine DDS : **DiscoveredWriterData**, **DiscoveredReaderData** et un troisième, optionnel, non considéré par les implémentations DDS. Il s'agit du topic **DiscoveredTopicData** que nous ignorerons dans la suite du texte. Le premier définit

TABLE 2.1 – Attributs du topic SPDPdiscoveredParticipantData

| **Attributs** | **Commentaires** |
|---|---|
| guidPrefix | Identifiant RTPS (*Globally Unique Identifier (GUID)*) du participant. Etant structurés de manière hiérarchique, les identifiants GUID des *endpoints* du participant sont déterminés en y concaténant un suffixe propre à chaque *endpoint* |
| vendorId | identifiant de l'implémentation du middleware DDS du participant |
| metatrafficUnicastLocatorList | Liste de localisateurs (combinaison d'adresse, numéro de port,type de protocole de transport) unicasts qui peut être utilisée pour joindre les *built-in* ***endpoint*** (i.e. *Data Readers*) du participant |
| metatrafficMulticastLocatorList | Liste des localisateurs multicasts qui peut être utilisée pour joindre les *built-in* ***endpoint*** du participant |
| defaultUnicastLocatorList | Liste, par défaut, de localisateurs unicasts qui peut être utilisée pour joindre des *endpoints* "utilisateur" (créés par l'application) du participant. Cette liste est prise en compte dans le cas où un *endpoint* ne spécifie pas explicitement ses propres adresses unicasts pour le joindre. |
| defaultMulticastLocatorList | finalité similaire à *defaultUnicastLocatorList* en considérant des adresses multicasts |
| availableBuiltinEndpoints | Permet de spécifier la liste des *built-in endpoints* disponibles dans le participant. Les valeurs des éléments de la liste correspondent aux différents types de *endpoints*, à savoir : PUBLICATIONS_READER, PUBLICATIONS_WRITER, SUBSCRIPTIONS_READER, SUBSCRIPTIONS_WRITER, TOPIC_READER, TOPIC_WRITER |
| leaseDuration | Durée de vie du participant en l'absence de réception des données périodiques relatives au participant. |

le contenu des messages SEDP émis par un *endpoint* de type *Data Writer* pour déclarer sa présence et ses caractéristiques. Comme l'indique la figure 2.3 [3], il étend le topic *PublicationBuiltinTopicData* (qui est un renommage RTPS du topic DDS *DCPSPublication* avec une correspondance exacte de leurs attributs) qui intègre les informations relatives au type de données qu'il produit (i.e. nom du topic et le nom du type) ainsi que les informations sur la QoS qu'il est capable de fournir (avec les différents paramètres de QoS de DDS). Ce sont les informations qui permettent de mettre en évidence la compatibilité entre un *Data Writer* avec un *Data Reader*. D'autre part, le topic *DiscoveredWriterData* étend le topic *WriterProxy* qui contient toutes informations permettant de localiser le *Data Writer* avec une liste de localisateurs unicasts et une liste de localisateurs multicasts. Le deuxième concerne les messages SEDP émis par des *Data Readers* pour déclarer leur présence et exprimer leur souhait de souscrire à un topic donné avec une QoS requise. Comme pour le topic précédent, le topic *DiscoveredReaderData* intègre les informations de localisation du *Data Reader*.

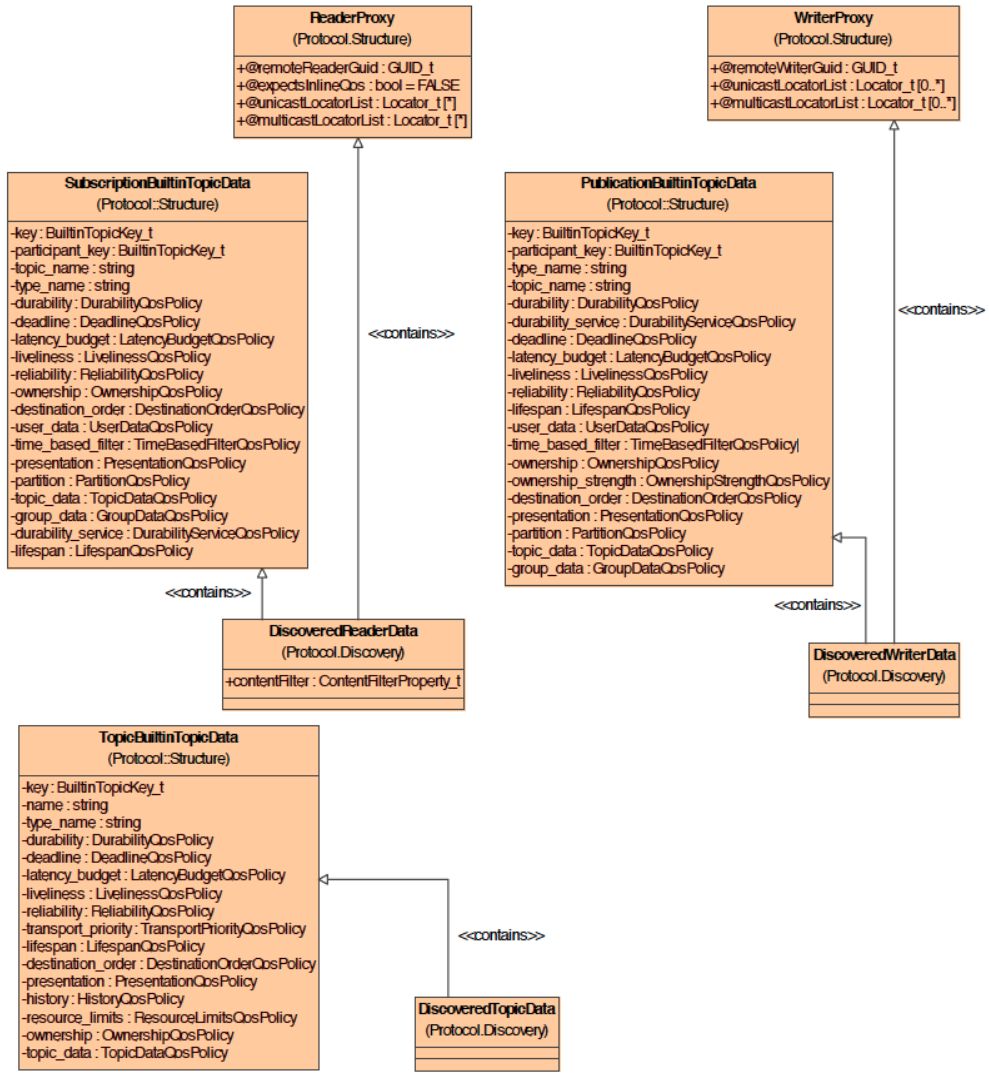


FIGURE 2.3 – Built-in topics définis par SEDP

Quatre *built-in endpoints* peuvent être intégrés à un participant lors de sa création : un *Reader* et un *Writer* pour chacun des deux topics décrits ci-avant (voir figure 2.4).

3. extraite du standard DDS relatif au protocole RTPS [18]

En effet, un participant se doit d'inclure les *built-in endpoints* nécessaires pour déterminer les correspondances entre les entités applicatives locales du participant avec les entités applicatives distantes. A titre d'exemple, si un participant DDS ne fait que produire des données dans le domaine (i.e. ne comporte que des *Data Writers* applicatifs), le participant n'est tenu à intégrer que le *built-in Data Writer* relatif au topic **DiscoveredWriterData** et le *built-in Data Reader* relatif au topic *DiscoveredReaderData*.

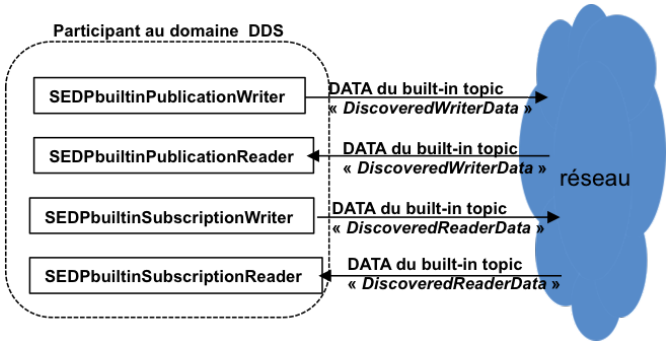


FIGURE 2.4 – built-in endpoints de SEDP

### 2.2.2 Méthode de capture des besoins applicatifs

Nous nous focaliserons sur la capture des besoins applicatifs relatifs à un domaine DDS donné. Notre algorithme exploite le service de découverte DDS basé sur le protocole SDP (seul protocole imposé par le standard DDS et celui qui est donc intégré à toutes les implémentations du middleware DDS) pour découvrir et référencer, dans le temps, l'ensemble des *Data Readers* et les *Data Writers* disponibles à un instant donné sur le domaine, avec pour chacun, le topic qui le concerne, la QoS associée à la distribution des données et sa localisation sur le réseau. En se référant à la section précédente, notre algorithme inclut un participant DDS qui intègre les *built-in endpoints* suivants : *SPDPbuiltinParticipantWriter* et *SPDPbuiltinParticipantReader* pour la découverte des participants au domaine et *SEDPbuiltinPublicationReader* et *SEDPbuiltinSubscriptionReader* pour la découverte des *Data Readers* et les *Data Writers* du domaine. Tous sont accessibles via l'API DDS, ce qui permet à notre algorithme d'établir pour chaque topic, les *Data Writers* et *Data Readers* impliqués dans le domaine avec leur QoS et leur localisation.

Le diagrame de séquence de la figure 2.5 décrit les interactions de l'algorithme de capture avec son environnement. Les premières interactions décrivent l'accès aux *built-in Data Readers*. Les interactions suivantes illustrent la découverte de l'arrivée d'un nouvel *Data Reader* puis la découverte d'un changements de la QoS associée au *Data Reader*.

## 2.3 Méthode de re-traduction des besoins de l'application en une demande de réseau virtuel

Le service réseau à déployer pour les applications d'un domaine DDS est un réseau Overlay constitué de plusieurs liens virtuels de bout-en-bout (de hôte terminal à hôte terminal) qui peuvent être d type point-à-point ou point-multipoints avec une capacité

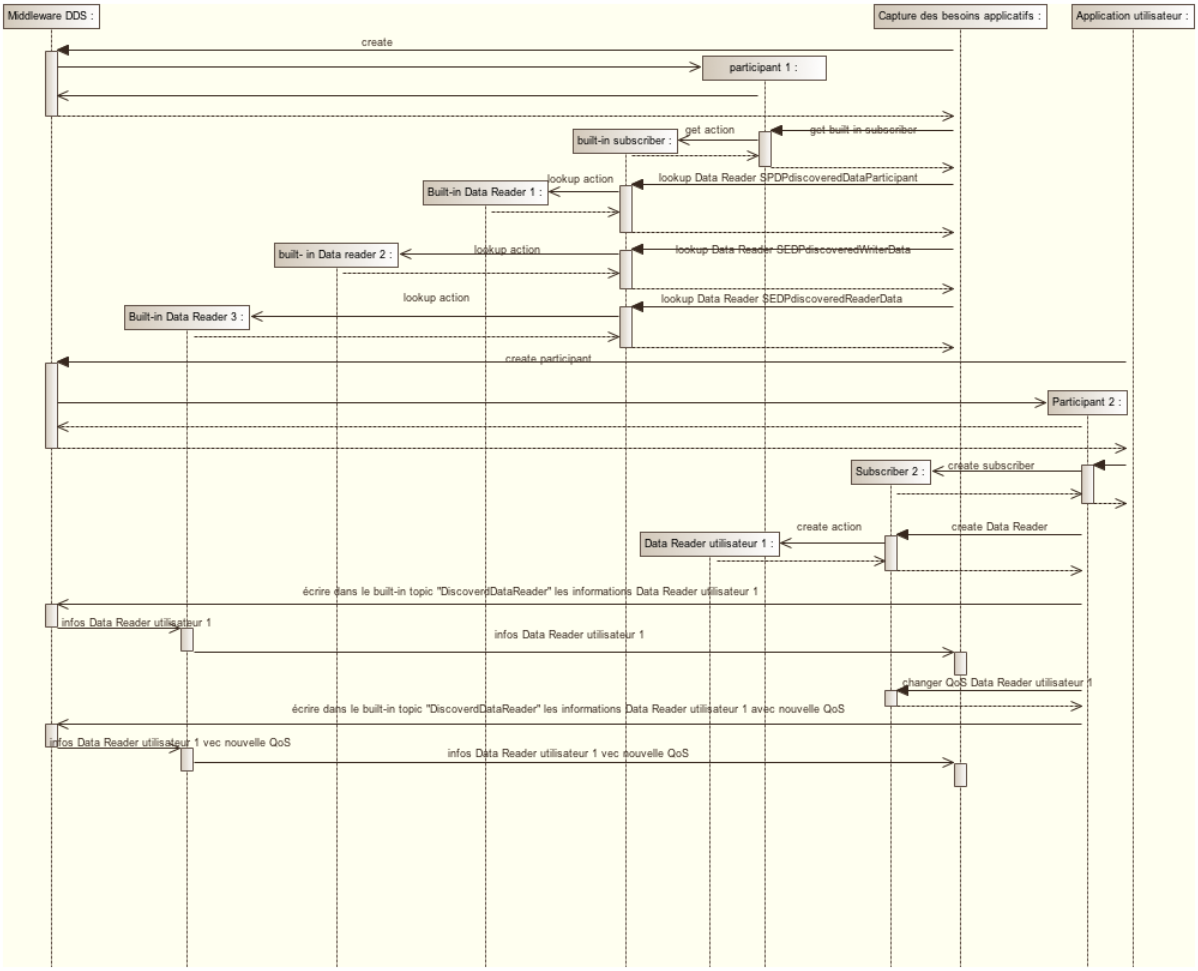


FIGURE 2.5 – Illustration des interactions de l'algorithme de capture des besoins

requise et un délai de transfert maximal à respecter. Nous décrivons dans ce qui suite la méthode pour passer de la liste des topics, de la section précédente, vers cette demande de réseau overlay. Pour des raisons de présentation, nous supposerons dans la suite que les données d'un topic (ou une instance de topic définie par une valeur de l'attribut *clef* du topic ) sont produites par un seul *Data Writer* mais potentiellement consommées par plusieurs *Data Readers*[4]. Sous cette hypothèse, nous faisons correspondre à chaque topic (ou instance de topic) un lien virtuel de bout-en-bout élémentaire avec les caractéristiques suivantes :

- extrémité source est l'adresse de localisation du *Data Writer* et le(s) extrémité(s) destination sont le(s) adresse(s) de localisation de(s) *Data Reader(s)*. Dans le contexte d'une architecture TCP/IP, ces adresses de localisation sont des adresses de niveau transport définies par une adresse IP et un numéro de port et le protocole de niveau transport utilisé (par défaut, UDP) ;
- la capacité du lien est déduite du paramètre de QoS DDS *Deadline* côté *Data Writer* qui spécifie la durée maximale entre deux écritures successives de l'application ce qui permet de déduire le débit minimum à garantir pour l'application. Cette approche a néanmoins quelques limites : D'une part, elle n'est efficace que si le trafic généré par l'application est plutôt régulier (dans le cas contraire, il

4. le cas multi-producteurs est traité en considérant un producteur à la fois

majorerait de manière pessimiste le trafic généré par l'application) ; Pour pallier à cette limite, il serait possible d'envisager dans la suite des travaux, d'appliquer des techniques d'estimation de trafic de l'application pour déduire la capacité du lien. D'autre part, si des *Data Readers* ont des besoins de QoS différents, ce sont les besoins les plus strictes qui s'appliqueront à tous, occasionnant un gaspillage de ressources. Adresser cette limite peut également faire l'objet de nos travaux futurs.

— Le délai de transfert du lien est déduit du paramètre de QoS DDS *Latency_ Budget* qui permet à l'application de spécifier une exigence sur la durée maximale entre le moment où un échantillon est écrit et le moment où il est placé dans la mémoire cache du DataReader.

Afin de réduire la complexité, en nombre de liens, du réseau virtuel à déployer et donc permettre à notre approche de mieux résister à l'échelle, une dernière étape cherche à établir des agrégations de liens virtuels élémentaires. La version initiale et actuelle de cet algorithme repose sur une technique d'agrégation simple : les liens élémentaires avec les mêmes extrémités sont agrégés. La capacité de l'agrégat étant fixé à la somme des débits de chaque lien élémentaire alors que le délai de transfert est fixé au minimum des délais de transfert des liens élémentaires.
La version actuelle de l'algorithme est clairement minimaliste et mériterait d'être exprimée selon un problème d'optimisation multi-critères : minimisation du nombre de liens tout en minimisation l'impact de l'agrégation sur les ressources gaspillées. Ce travail est envisagé dans la suite des travaux.

## 2.4 Conclusion

Nous avons présenté dans ce chapitre la méthode que nous utilisons pour capturer de manière précise les besoins applicatifs d'un domaine DDS. Pour des raisons de simplicité, nous avons fait le choix de l'implémenter comme module externe au middleware DDS. Ce choix implique un coût (sur-débit pour délivrer les données des participants DDS, Data Readers et Data Writers) jusqu'à notre composant, mais ce dernier demeure limité. Nous avons également décrit la méthode que nous utilisons pour retraduire les besoins applicatifs en une requête de réseau virtuel constitué de plusieurs liens virtuels de bout-en-bout. Nous avons mis en avant la nécessité d'une étape d'agrégation de ces liens virtuels pour réduire leur nombre avec une première méthode simple d'agrégation. Ces algorithmes ont été codés et sont fonctionnels [24].

Les perspectives du travail présenté dans ce chapitre sont : Analyser plus finement l'intérêt (et les gains) d'intégrer la phase de capture des besoins au middleware DDS ; Proposer une version plus élaborée de l'algorithme d'agrégation des liens.

# Chapitre 3

# L'algorithme d'allocation de ressources

## 3.1 Problèmes d'allocation de ressources dans un contexte de virtualisation réseau

### 3.1.1 L'allocation de ressources pour des réseaux virtuels

L'allocation de ressources pour des réseaux virtuels est un problème NP-Difficile [10] qui consiste à allouer un réseau virtuel $R_V$ sur un réseau physique $R_P$. Cette allocation consiste à trouver un placement $M$ des routeurs et liens virtuels, tel que :

- Chaque routeur (nœud) virtuel $n^v$ soit alloué sur un routeur (nœud) physique $n^p = M(n^v)$.
- Chaque lien virtuel $l^v$ reliant deux nœuds virtuels $n_1^v$ et $n_2^v$ soit alloué sur un chemin $M(l^v) = (l_1^p, l_2^p, \ldots, l_k^p)$ reliant $M(n_1^v)$ et $M(n_2^v)$.

Comme expliqué précédemment, le placement $M$ doit en général respecter un ensemble de contraintes en termes de QoS, performances et sécurité. Les différentes contraintes pouvant être concernées par l'allocation sont définies dans la section 3.1.4.

Ainsi, l'allocation de ressources pour de la virtualisation réseaux met en jeu deux problèmes d'allocation interdépendants : allocation de nœuds virtuels à des nœuds physiques et allocation de liens virtuels à des chemins physiques.

Un nœud virtuel ne peut être distribué sur plusieurs nœuds physiques, et pour une requête de virtualisation donnée, un seul nœud virtuel peut être alloué sur chaque nœud physique.

Un lien virtuel peut avoir un ensemble de nœuds destination $n_{d_1}^p, n_{d_2}^p, \ldots, n_{d_k}^p$ (liens de type multicast), ou un seul (lien de type unicast). De plus, il peut être alloué sur différents chemins physiques (utilisation du *path-splitting*).

Dans la suite de ce travail, nous nous sommes plus particulièrement centré sur l'allocation de liens virtuels.

### 3.1.2 Cas particulier de l'allocation de ressources pour des liens virtuels

L'allocation de liens virtuels trouve sont intérêt auprès de fournisseurs de services proposant la création de « tunnels » entre différentes destinations, avec des garantie en termes de bande passante, QoS, sécurité et performances.

L'allocation de ressources pour des liens virtuels est un sous-problème de l'allocation de ressources pour des réseaux virtuels dans lequel les nœuds virtuels sont déjà placés sur le réseau physique. On cherche donc à trouver un ou plusieurs chemins entre plusieurs routeurs physiques (une source $n_s^p$ et des destinations $n_{d_1}^p, n_{d_2}^p, \ldots, n_{d_k}^p$), sous différentes contraintes (cf. section 3.1.4).

### 3.1.3 Les requêtes

Dans un problème d'allocation de ressources pour des réseaux virtuels, les requêtes sont des demandes d'allocation d'un ou plusieurs réseaux ou liens virtuels. Elles peuvent être très variées et être plus ou moins contraintes (voir 3.1.4).

L'allocation de ressources peut intervenir :

1. A chaque arrivée de requête : On cherche alors à déterminer puis affecter les ressources nécessaires pour répondre à celle-ci sans remettre en cause les ressources déjà allouées sur le réseau physique pour les requêtes précédentes. Ce cas de figure (que nous considérerons ici) trouve son application dans le cadre d'un fournisseur de réseaux virtuels à la demande avec un déploiement de quelques minutes.
2. En connaissant préalablement toutes les requêtes de réseaux virtuels à satisfaire : On peut alors tirer profit de cette connaissance globale pour obtenir une meilleure allocation. Ce cas de figure intervient dans des phases de dimensionnement et d'optimisation du réseau. Ce modèle de fourniture de réseaux virtuels est beaucoup moins flexible que le précédent.

### 3.1.4 Les différents aspects des problèmes d'allocation de ressources pour réseaux virtuels

Les problèmes d'allocation de ressources pour les réseaux virtuels peuvent prendre plusieurs formes ou s'exprimer de différentes manières selon les contraintes et objectifs visés.

#### 3.1.4.1 La topologie du réseau

Dans les problèmes de virtualisation, l'une des données les plus importantes est la topologie des réseaux virtuels et physiques : les routeurs et les liens. En général, on représente une topologie réseau sous forme d'un graphe orienté $G = (V, E)$ où $V$ représente l'ensemble des routeurs (nœuds) et $E \subseteq V \times V$ l'ensemble des liens.

#### 3.1.4.2 La bande passante exigée

La bande passante est certainement l'aspect le plus important à prendre en compte lors des problèmes d'allocation de ressources pour réseaux virtuels et liens virtuels. Il est

nécessaire de trouver un chemin physique reliant deux nœuds avec une capacité suffisante par rapport aux demandes, c'est à dire :

— Respectant les contraintes de capacité au niveau des liens physiques formant le chemin.
— Respectant les contraintes de conservation du flux au niveau des routeurs composant le chemin physique.

En plus de ces deux contraintes fondamentales sur la bande passante, d'autres caractéristiques peuvent être prises en compte, notamment :

— Le **Path Splitting** - Distribuer le trafic correspondant à un lien virtuel (une source et une destination) sur plusieurs chemins physiques. Cette caractéristique requiert des protocoles spécifiques au niveau du réseau (le besoin d'étiqueter certains paquets, entre autres).
— Le **Multicast** - Envoyer un message d'une source vers $m$ destinations, sans avoir besoin de créer $m$ chemin disjoints.

Lors de la définition des requêtes de virtualisation, il est possible de considérer soit le débit moyen demandé par la requête, soit le débit maximal. Dans le premier cas, le réseau virtuel sera alors surdimensionné, dans le deuxième cas il y des risques de congestion au niveau des routeurs.

#### 3.1.4.3 Les routeurs

**3.1.4.3.1 Couts liés à la virtualisation des routeurs** Dans un contexte d'allocation de réseaux virtuels, un des aspects importants à prendre en compte est la charge des routeurs physiques sur lesquels sont alloués des routeurs virtuels : un routeur physique ne peut supporter qu'un nombre fini de routeurs virtuels dépendant de sa capacité et des besoins des composants virtuels. La charge d'un routeur liée à la virtualisation peut apparaître sous différentes formes : Charge processeur, RAM, etc. En réalité, il est aujourd'hui difficile de quantifier les ressources utilisées par la virtualisation d'un routeur [21].

**3.1.4.3.2 Couts liés à la commutation de paquets** À la fois dans un contexte d'allocation de réseaux et de liens virtuels, la charge des routeurs liée à la commutation des paquets est un élément non négligeable à prendre en compte pour la résolution du problème de virtualisation. Dans un contexte SDN / OpenFlow [11, 16], la charge de commutation d'un routeur est étroitement liée au nombre de règles traitées par le routeur (i.e. la taille de la table des flots). Les scénarios de routage étant très variés (multicast, path-splitting, etc.), il est actuellement difficile de quantifier de façon générique le nombre de règles nécessaire à la réalisation du routage d'un lien virtuel.

**3.1.4.3.3 La localisation** Dans certains problèmes, il peut être nécessaire d'ajouter des contraintes de localisation sur les nœuds virtuels. Elles peuvent être de différents types :

— Des contraintes sur la distance entre les positions physiques des éléments (exemple : deux routeurs physiques sur lesquels sont alloués deux routeurs virtuels reliés par un lien virtuel ne peuvent être distants de plus de 200 mètres) - Cette contrainte peut être utile lors de l'allocation sur des réseaux composés de plusieurs ensembles de nœuds (bâtiments / pièces), afin de forcer l'emplacement de plusieurs

routeurs virtuels dans des endroits peu distants (pour des questions logistiques, par exemple).
— Des contraintes de placement pour les nœuds virtuels : un nœud virtuel ne peut être placé que dans une zone prédéfinie (ou un ensemble spécifique de nœuds physiques).
— Des contraintes d'allocation : certains nœuds virtuels sont déjà placés sur le réseau physique.

#### 3.1.4.4 Délais de transfert des paquets

Le délai définit le temps nécessaire pour qu'un paquet circule d'un routeur physique $n_1$ vers un routeur physique $n_2$. La contrainte en général exprimée est celle du « délai admissible », i.e le délai maximal qu'un paquet ne doit pas dépasser. C'est une contrainte importante en terme de QoS, et donc un aspect très important dans les problèmes de virtualisation. Le délai peut être pris en compte à différents endroits :

— Sur les liens, via le temps de transmission d'un message (correspondant au temps d'émission du message par un routeur plus le temps de propagation du message sur le lien).
— Sur les routeurs, en fonction de la congestion de ceux-ci (cf. flux moyen / max) et du temps de traitement des paquets (dépendant de la charge des routeurs).

#### 3.1.4.5 La migration des allocations

La migration est un aspect mis de plus en plus en avant dans les réseaux reconfigurables autorisant la réallocation de requêtes de virtualisation déjà placées afin de permettre le traitement d'une nouvelle requête.

En général, la migration consiste à identifier les éléments critiques du réseaux avant d'essayer de les soulager sans remettre en cause les besoins et contraintes des requêtes déjà traitées.

La migration possède en général un coût non négligeable, à la fois au niveau du traitement des requêtes (algorithme plus lourd car plus libre) et au niveau des réaffectations (temps de migration des ressources d'un élément vers un autre) alors que le réseau virtuel est en phase opérationnelle. [13]

#### 3.1.4.6 Objectifs

**3.1.4.6.1 Équilibrage des charges** Lors de la résolution du problème d'allocation de réseaux ou liens virtuels, un des objectifs fréquemment considéré consiste à rechercher un équilibrage de l'utilisation du réseau physique. Cette équilibrage consiste à préférer distribuer la bande passante sur plus de liens physiques, plutôt que surcharger un nombre restreint de liens. On peut également vouloir équilibrer la charge des routeurs intermédiaires, afin d'éviter toute congestion.

**3.1.4.6.2 Coûts et revenus financiers** Suivant le contexte, l'aspect financier lié à l'allocation de réseaux et de liens virtuels peut être intégré au problème. Il peut apparaître sous 2 formats : Le coût et le revenu.

Le coût est principalement lié à l'utilisation des composants physiques du réseau (lien, routeur, etc.).

Le revenu est dépendant de la requête. Dans un concept de fournisseur, une requête avec un revenu élevée peut être vue comme prioritaire par rapport aux autres. Dans ce même concept, la satisfaction d'un maximum de requête (le *taux d'acceptation*) est un enjeu important.

## 3.2 État de l'art de la littérature existante

La plupart des travaux existants de la littérature actuelle peuvent être séparés en deux familles : ceux traitant de l'**Allocation de réseaux virtuels** (*Virtual Network Mapping / Embedding*) dans lesquels on considère de manière simultanée les deux problèmes d'allocation de nœuds et de liens, et ceux traitant de l'**Allocation de liens virtuels** dans lesquels on ne s'intéresse qu'à l'allocation des liens (les extrémités de ces liens étant déjà placés sur le réseau physique).

### 3.2.1 L'allocation de ressources pour des réseaux virtuels

L'allocation de réseaux virtuels a récemment suscité un engouement dans la communauté scientifique qui a mené à l'apparition de différentes méthodes de résolution.

Dans [15], Nogueira et al. proposent un **algorithme polynomial** pour l'allocation de réseaux virtuels basé sur une **heuristique** et utilisant la notion de *stress*. Le stress est une valeur propre à chaque élément physique dépendant des ressources disponibles lors de la requête (Bande passante utilisée et disponible pour les liens, **CPU** / **RAM** pour les routeurs).

Dans [10], Lischka and Karl utilisent un algorithme de **recherche de sous-graphe isomorphisme** pour trouver une allocation possible d'un réseau virtuel sur un réseau physique. Leur algorithme prend en compte la **charge des routeurs et des liens** ainsi qu'un **délai** calculé par rapport au nombre de sauts sur un chemin physique (nombre de liens physiques utilisés pour créer un lien virtuel).

Hsu et al. proposent dans [7] différents algorithmes pour l'allocation de routeurs et liens virtuels avec possibilité de **path-splitting et migration**. L'algorithme de path-splitting proposé est récursif et n'utilise le path-splitting que si nécessaire : pour un besoin $B$ en bande passante, l'algorithme va d'abord chercher un chemin de capacité au moins égale à $B$, s'il ne trouve pas, il tente de trouver deux chemins de capacité au moins égale à $B/2$ (avec des arcs en commun de capacité au moins égale à $B$), et ainsi de suite. Si l'algorithme ne parvient pas à trouver un chemin après plusieurs itérations, il va alors chercher à migrer des chemins existants. Les résultats de Hsu et al. montrent l'intérêt du path-splitting et de la migration dans les problèmes d'allocation de réseaux virtuels. Cette approche utilisant le path-splitting et la migration est également abordée par Yu et al. dans [28] qui propose une méthode de type « Try and Error » : Allocation des nœuds, tentative d'allocation des liens sans path-splitting, tentative avec path-splitting si les premières ont échoué (basée sur la résolution d'un problème de *Multi Commodity Flow*), migration des nœuds si nécessaires et enfin migrations des liens (changement de ratio dans les splits ou ré-allocation totale) si aucun des essais précédents n'a abouti. L'approche de Yu et al. diffère en revanche de celle de Hsu et al. dans la gestion de l'arrivée des requêtes : Alors que Hsu et al. traitent les requêtes une par une, Yu et al.

attendent d'avoir un certains nombre de requêtes avant de les allouer dans le but de commencer par celles offrant le meilleur revenu.

Les deux articles [1] (réédité en 2012, voir [2]) et [14] utilisent une formulation de **Programmation Linéaire en Nombres Entiers** pour tenter de résoudre le problème. Dans [14], Melo et al. proposent une résolution du problème sans relaxation, avec prise en compte de plusieurs contraintes (**Bande passante**, **délai**, **position des routeurs**, **charge des routeurs (CPU / RAM)**), et des résultats avec différentes fonctions objectives. Chowdhury et al. [1] proposent une résolution utilisant une relaxation du problème : la relaxation permet de trouver rapidement un placement des routeurs. Deux versions sont proposées : la première alloue chaque routeur virtuel $n^v$ sur le routeur physique $n^p$ possédant le meilleur score retourné par la résolution du problème relaxé, la deuxième ajoute de l'aléatoire dans le choix des routeurs. Les chemins sont ensuite alloués via un algorithme de *Multi Commodity Flow*.

### 3.2.2 L'allocation de ressources pour des liens virtuels

L'article [6] propose une méthode de **résolution par contraintes** (CSP) ne prenant en compte que les besoins en **bande passante** des requêtes. Les routeurs sont regroupés pour former des « Blocking Islands », permettant ainsi l'utilisation d'heuristiques efficaces pour le choix des variables et des valeurs à affecter durant la résolution du CSP. Frei and Faltings montrent que leur méthode permet d'éviter l'allocation de liens critiques lorsque ce n'est pas nécessaire, contrairement à des algorithmes de plus court chemin classiques.

Le problème de l'allocation de ressources pour liens virtuels peut facilement s'assimiler à un problème de *Multi Commodity Flow*. Masri et al. [12] proposent un algorithme basé sur une méta-heuristique *Colonie de Fourmis* pour résoudre le problème du routage de plusieurs flots multisource et multicast avec prise en compte de la bande passante et minimisation du délai et du coût. Wei et al. proposent dans [26] un algorithme basé sur la prédiction du trafic dans le réseau et utilisant les algorithmes de *Djikstra* et de résolution d'un *Multi Commodity Flow* : L'algorithme tente d'abord de résoudre le problème en utilisant l'algorithme de Djikstra, puis un algorithme de résolution de *Multi Commodity Flow* si le premier n'a pas abouti.

Xi and Yeh proposent dans [27] un algorithme distribué pour le routage de flots multicast. L'algorithme trouve d'abord un chemin de la source vers chaque destination $t$, utilisant sur chaque lien $(i, j)$ une quantité de bande passante $b_t(i, j)$. La quantité de bande passante consommée $b(i, j)$ par le lien multicast peut alors être calculée via $b(i, j) = \max b_t(i, j)$. Kucharzak and Walkowiak proposent dans [9] différentes heuristiques pour l'allocation d'arbres **multicast** prenant en compte uniquement des contraintes au niveau de la bande passante.

### 3.2.3 Synthèse

Le tableau 3.1 fournit un récapitulatif des différents aspects pris en compte dans chaque article cité dans la section précédente.

Les caractéristiques et contraintes considérées par les travaux de la littérature actuelle sont variés. Alors que tous prennent en compte les besoins en bande passante, seulement quelques un prennent en compte le délai, ou la charge des routeurs. La plupart des

TABLE 3.1 – Récapitulatif des différents types d'allocation et contraintes étudiés par chaque article

| | Allocation | Contraintes | Type de liens | Techniques |
|---|---|---|---|---|
| Masri et al. [12] | Liens | Bande Passante<br>Délai | Multi-sources<br>Multicast | |
| Nogueira et al. [15] | Réseaux | Bande Passante<br>Charge routeurs (CPU / RAM) | Unicast | |
| Lischka and Karl [10] | Réseaux | Bande Passante<br>Délai<br>Charge routeurs (CPU / RAM) | Unicast | |
| Hsu et al. [7] | Liens | Bande Passante | Unicast | Path-Splitting<br>Migration |
| Chowdhury et al. [1] | Réseaux | Bande Passante<br>Localisation des routeurs<br>Charge routeurs (CPU / RAM) | Unicast | |
| Melo et al. [14] | Réseaux | Bande Passante<br>Délai<br>Localisation des routeurs<br>Charge routeurs (CPU / RAM) | Unicast | |
| Frei and Faltings [6] | Liens | Bande Passante | Multicast | |
| Wei et al. [26] | Réseaux | Bande Passante | Unicast | |
| Hsu et al. [7] | Réseaux | Bande Passante<br>Localisation des routeurs | Unicast | Path-Splitting<br>Migration |
| Xi and Yeh [27] | Liens | Bande Passante | Multicast | |
| Kucharzak and Walkowiak [9] | Liens | Bande Passante | Multi-sources | |

algorithmes proposés ne font pas usage du multicast, du path-splitting ou de la migration pour obtenir des meilleurs résultats. Finalement, aucun ne prend en compte la charge des routeurs liée à la commutation de paquets.

## 3.3 Positionnement de notre problème

### 3.3.1 Définition

#### 3.3.1.1 Le problème

Dans la suite, nous nous intéressons aux problèmes d'**allocation de liens virtuels**, avec prise en compte des contraintes de **bande passante** (avec possibilité de path-splitting), de **délai** et de **charge des routeurs liée à la commutation de paquets**.

Bien que la plupart de ces aspects aient déjà été étudiés de manières isolés dans de

nombreux travaux, il n'existe pas à notre connaissance de méthode pour l'allocation de réseaux ou liens virtuels les incluant tous. De plus, aucun article à ce jour ne prend en compte la charge des routeurs liées à la commutation dans les problèmes d'allocation de réseaux virtuels.

#### 3.3.1.2 Les requêtes

On considérera des requêtes arrivant les unes après les autres, sans remise en cause de l'allocation des requêtes précédentes. Chaque requête sera caractérisée par un temps d'arrivée et une durée de vie.

Les requêtes d'allocation traitées seront composées d'une ou plusieurs *sessions*. Chaque *session* représente un lien virtuel possédant une source unique (un routeur physique) et un ensemble de destinations (routeurs physiques), permettant ainsi des liens point à point et point à multipoint. Une session est caractérisée par une demande identique en termes de bande passante et de délai à respecter.

De plus, afin de limiter une trop grande fragmentation liée à l'utilisation du path-splitting, nous ajoutons à chaque session une valeur minimale de bande passante. En effet, il peut être préférable, pour une session demandant une bande passante de 1 Gbps, de ne pas allouer des quantités de bande passante inférieures à 200 Mbps sur un lien physique.

#### 3.3.1.3 Les allocations possibles

En fonction de ses caractéristiques, un lien virtuel pourra être alloué sur le réseau physique en utilisant :

— Un seul chemin : lien virtuel point à point (unicast), sans « path-splitting ».
— Un arbre : lien virtuel point à multipoint (multicast), sans « path-splitting ».
— Un ensemble de chemins : lien virtuel point à point (unicast) ou point à multipoint (multicast), avec « path-splitting ».

### 3.3.2 Hypothèses

Dans la suite, on se placera dans l'hypothèse d'un réseau entièrement virtualisé où l'utilisation d'une ressource physique est bornée par sa capacité, i.e. elle ne peut pas être allouée à une requête si celle-ci a un besoin supérieur à la capacité restante.

#### 3.3.2.1 Perte de paquets et délai lié à la congestion

Sous cette hypothèse, on peut considérer qu'il n'y a pas de congestion au niveau des routeurs (l'utilisation des liens entrants ne peut être supérieure à celle des liens sortants), et donc qu'il n'y a pas de perte de paquets ou de délai liés à la congestion des routeurs.

#### 3.3.2.2 Charge des routeurs

On se place dans le cadre d'un réseau programmable (*Software Defined Network*), de type *OpenFlow* par exemple.

Comme indiqué précédemment, il est difficile de quantifier la charge des routeurs liées à la commutation, c'est pourquoi l'hypothèse suivante sera faite : la charge d'un routeur liée à la commutation de paquets est directement proportionnelle au nombre d'entrées dans sa table des flots, elle même proportionnelle au nombre de flots entrants dans ce routeur. Ainsi la charge d'un routeurs sera proportionnelle au nombre de liens entrants dans ce routeur.

#### 3.3.2.3 Full-Duplex

On considérera que l'intégralité des liens du réseau physique fonctionnent en mode full-duplex, c'est à dire permettant les communications dans les deux sens **simultanément**. Ainsi un lien $(i, j)$ de capacité 100 Gbps possède des ressources pouvant soutenir un lien virtuel de capacité 100 Gbps de $i$ vers $j$ ET un lien virtuel de même capacité de $j$ vers $i$. Les ressources utilisées dans un sens n'entrent donc pas en compte dans le calcul des ressources utilisées dans l'autre sens.

## 3.4 Modélisation

Après avoir défini le problème de virtualisation considéré dans le chapitre précédent, ce chapitre propose une modélisation non linéaire en utilisant le formalisme de la programmation mathématique.

### 3.4.1 Données en entrée

#### 3.4.1.1 Le réseau physique

La topologie du réseau physique est représentée par un graphe $G = (V, E)$ où $V$ ($|V| = N$) est l'ensemble des nœuds physiques et $E$ ($|E| = M, E \subseteq V \times V$) l'ensemble des liens physiques reliant deux nœuds.

On notera sans distinction $(i, j)$ ou $e$ un lien physique, en fonction du contexte, et on considérera que tous les liens sont bidirectionnels (full-duplex).

Chaque nœud est défini par sa capacité (nombre d'entrées acceptables dans la table de flots) $u_i$ et chaque lien est défini par sa capacité $c_{ij}$ et son temps de propagation $d_{ij}$.

$$\forall (i, j) \in E : d_{ij} = d_{ji} \; et \; c_{ij} = c_{ji} \tag{3.1}$$

On définit la fonction $Ng : V \to 2^V$ qui associe à un nœud $i$ donné sa liste de voisins, i.e. :

$$Ng(i) = \{j | j \in V, (i, j) \in E \; ou \; (j, i) \in E\}$$

#### 3.4.1.2 Les requêtes

On suppose en entrée un ensemble de $K$ sessions unicast ou multicast $k_1, \ldots, k_K$, chaque session étant définie par :

— Une source $s_k \in V$.

— Un ensemble de destination(s) $T_k \subseteq V - \{s_k\}$. Dans le cas d'un lien unicast, $|T_k| = 1$.
— Un besoin en bande passante $b_k \in \mathbb{N}^*$ et un délai acceptable $d_k \in \mathbb{N}^*$.
— La taille maximale des paquets $p_k$ transmis par la session.
— La bande passante minimale $b_k^{min}$ utilisable sur un lien (pour la gestion du « Path-Splitting » afin d'éviter un trop gros découpage dans les flux).

**Remarque :** Si $b_k^{min} = b_k$, le flux associé à la session ne peut pas être divisé et réparti sur différents chemins. En pratique, on définira $b_k^{min}$ comme un pourcentage (ratio) de $b_k$, $b_k^{min} = PS_{ratio} * b_k$ avec $PS_{ratio} \in [0, 1]$ (Il y a possibilité de path-splitting si $PS_{ratio} \leq 0.5$, pour éviter des calculs inutiles on fixera $PS_{ratio} = 1.0$ lorsque que le path-splitting n'est pas autorisé).

#### 3.4.1.3 Évolution temporelle

En pratique, on considérera des requêtes arrivant les unes à la suite des autres, chacune étant traitée indépendamment des autres (i.e. chaque arrivée de requête donne lieu à une nouvelle résolution du modèle, puisque le modèle est construit pour l'allocation d'une seule requête).

Lors de l'arrivée d'une requête au temps $t$, afin de prendre en compte les précédentes allocations, il est nécessaire de connaître les ressources du réseaux utilisées au temps $t$, soit :

— La charge actuelle d'un routeur $i$ : $\overline{u_i}$
— La charge actuelle d'un lien $e$ : $\overline{c_e}$

### 3.4.2 Variables

#### 3.4.2.1 Flux

Pour chaque session $k \in K$ et chaque destination $t \in T_k$, on définit une fonction $f_t^k : E \to \mathbb{N}$ telle que $f_t^k(i, j)$ définit la quantité de bande passante consommée sur $(i, j)$ pour faire transiter des données de $s_k$ vers $t$.

La quantité réelle de bande passante utilisée pour une session $k \in K$ sur un lien physique $(i, j) \in E$, $f^k(i, j)$ est alors définie par :

$$f^k(i, j) = \max_{t \in T_k} f_t^k(i, j) \tag{3.2}$$

#### 3.4.2.2 Délai

A nouveau, pour chaque session $k \in K$ et chaque destination $t \in T_k$, on définit une fonction $d_t^k : V \to \mathbb{N}$ tel que $d_t^k(i)$ représente le délai maximum pour qu'un message à destination du nœud destination $t \in T_k$ atteigne le nœud $i$ en partant de la source $s_k$. Le calcul des $d_t^k(i)$ utilise la valeur de bande passante consommée sur les liens et est défini par 3.3.

$$\forall k \in K, \forall t \in T_k, \forall i \in V : d_t^k(i) = \begin{cases} 0 & si\ i = s_k \\ \max\limits_{\substack{j \in Ng(i) \\ f_t^k(j,i) > 0}} (d_t^k(j) + \frac{p_k}{f_t^k ij} + d_{ji}) & si\ i \neq s_k \end{cases} \quad (3.3)$$

Le délai $d_t^k(i)$ prend en compte l'ensemble des voisins du nœud $i$ ($j \in Ng(i)$) tels que le lien $(j, i)$ soit utilisé par la session $k$ pour la cible $t$ ($f_k^t(j, i) > 0$).

**Remarque :** Le délai calculé est une borne supérieure du délai réel, ce qui, dans certains cas, pourra entraîner le refus biaisé d'une requête et donc empêchera la résolution optimale du problème.

#### 3.4.2.3 Charge des routeurs

Pour chaque nœud $i \in V$ et chaque session $k \in K$, on associe une variable $c^k(i) \in \mathbb{N}$ définissant les ressources de commutation utilisées par la session $k$ sur le nœud $i$. Le calcul de la charge, sous l'hypothèse définie en 3.3.2.2 correspond donc au nombre d'entrées effectives dans le routeur et est donné par l'équation suivante :

$$\forall k \in K, \forall i \in V : c^k(i) = \left| \left\{ j | j \in Ng(i), f^k(j, i) > 0 \right\} \right| \quad (3.4)$$

### 3.4.3 Contraintes

#### 3.4.3.1 Conservation du flot et respect de la demande en bande passante

Comme dans un problème de *Multi-Commodity Flow* standard, on impose une contrainte sur le flux dans chaque nœud du réseau physique, spécifiant que la quantité de données entrantes est égale à la quantité de données sortantes (à l'exception des nœuds source et destinations).

$$\forall k \in K, \forall t \in T_k, \forall i \in V : \sum_{j \in Ng(i)} (f_t^k(i, j) - f_t^k(j, i)) = \begin{cases} b_k & si\ i = s_k \\ -b_k & si\ i = t \\ 0 & sinon \end{cases} \quad (3.5)$$

#### 3.4.3.2 Respect de la contrainte de capacité des liens physiques

Pour s'assurer que la capacité en bande passante de chaque lien physique est respectée, on pose la contrainte suivante :

$$\forall (i, j) \in E : \sum_{k \in K} f^k(i, j) \leq c_{ij} - \overline{c_{ij}} \quad (3.6)$$

**Remarque :** Cette contrainte n'est plus valable si certains liens du réseau physique ne sont plus en mode full-duplex.

#### 3.4.3.3 Respect de la contrainte de bande passante minimale

Pour s'assurer que si une partie du flux $f_t^k$, associé à la cible $t$ d'une session $k$ est alloué sur le lien $e$, alors la quantité allouée est au moins $b_k^{min}$, on ajoute la contrainte suivante :

$$\forall k \in K, \forall t \in T_k, \forall e \in E : f_t^k(e) \neq 0 \Rightarrow f_t^k(e) \geq b_k^{min} \tag{3.7}$$

#### 3.4.3.4 Respect de la contrainte de délai des sessions

Le respect de la contrainte de délai pour une session $k$ est définie par 3.8.

$$\forall k \in K, \forall t \in T_k : d_t^k(t) \leq d_k \tag{3.8}$$

### 3.4.4 Fonction(s) objective(s)

Lors de la résolution du problème de virtualisation considéré, nous allons chercher à la fois à équilibrer la charge du réseau physique mais aussi à limiter l'utilisation des ressources du réseau physique.

#### 3.4.4.1 Équilibre des charges

La première fonction objective considérée cherche à équilibrer la charge du réseau physique exprimée en termes de charge au niveau des routeurs et des liens. Pour exprimer cet objectif d'équilibrage de charge, deux versions de cette fonction objective sont proposées : une basée sur la minimisation de la somme des carrés (3.9), l'autre sur la minimisation du maximum (3.10). Chacune de ces versions est décomposée en deux parties : la première représente la charge des liens (pondérée par le coefficient $\alpha$) et la seconde représente la charge des routeurs (pondérée par le coefficient $\beta$).

$$\text{minimiser } \alpha * \sum_{e \in E} \left( \frac{1}{c_e} * \left( \overline{c_e} + \sum_{k \in K} f^k(e) \right)^2 \right) + \beta * \sum_{i \in V} \left( \frac{1}{u_i} * \left( \overline{u_i} + \sum_{k \in K} c^k(i) \right)^2 \right) \tag{3.9}$$

$$\text{minimiser } \alpha * \max_{e \in E} \left\{ \frac{1}{c_e} * \left( \overline{c_e} + \sum_{k \in K} f^k(e) \right) \right\} + \beta * \max_{i \in V} \left\{ \frac{1}{u_i} * \left( \overline{u_i} * \sum_{k \in K} c^k(i) \right) \right\} \tag{3.10}$$

La deuxième version proposée (3.10) à l'avantage d'être facilement linéarisable, mais perd très vite de son intérêt lorsqu'un des composants du réseau physique devient surchargé (le max vaut alors 1 et il n'y a plus de recherche d'équilibre).

#### 3.4.4.2 Minimisation du nombre de routeurs utilisés

Le deuxième fonction objective considérée cherche à minimiser le nombre de routeurs utilisés, dans l'optique de réduire la consommation d'énergie.

$$\text{minimiser} \left| \left\{ i | i \in V, \sum_{k \in K} \left( c^k(i) + \overline{u_i} \right) > 0 \right\} \right| \tag{3.11}$$

### 3.4.5 Variation sur le calcul de la fonction objective

Les valeurs principales utilisées par la fonction objective sont les $f_k(i)$ et $c_k(i)$.

Pour chacun des 2 types de valeurs, on peut considérer 3 variations au niveau des fonctions objectives :

1. Ne prendre en compte que les ressources utilisées par la requête actuelle pour chaque élément.
2. Prendre en compte les ressources utilisées par la requête ainsi que les ressources déjà consommées sur chaque élément, mais ne pas compter les éléments sur lesquels aucune ressource n'est consommée par la requête actuelle.
3. Prendre en compte tous les éléments, ainsi que les ressources déjà utilisées, même si aucune ressource n'est consommée par la requête actuelle sur l'élément.

Dans le cas 1, il suffit de poser $\overline{c_e} = 0$ et $\overline{u_i} = 0$ pour tout $e \in E$ et $i \in V$. Le cas 3 est celui présenté dans ce document. Le cas 2 s'obtient en remplaçant les ensembles d'itérations des sommes et maximums de la façon suivante :

$$E' = \left\{ e \in E, \sum_{k \in K} f_k(e) > 0 \right\} \text{ et } V' = \left\{ i \in V, \sum_{k \in K} c_k(i) > 0 \right\}$$

### 3.4.6 Apport du modèle par rapport à l'existant

Le modèle présenté ici permet l'allocation de ressources pour des liens virtuels sur un réseau physique. Bien qu'il existe déjà plusieurs modèles pour ce problème (voir chapitre 3), la particularité de celui-ci réside dans :

— La possibilité d'allouer des liens de type point à point ou point à multipoint (multicast).
— La prise en compte de contraintes QoS de type bande passante et délai au niveau des liens virtuels.
— La prise en compte au niveau des routeurs de la charge liée à la commutation des paquets.

## 3.5 Modèle linéaire

Le modèle proposé au chapitre précédent n'est pas linéaire, son efficacité pratique est ainsi limitée. Dans ce chapitre, nous allons présenter une linéarisation possible.

La linéarisation consiste à ramener chaque contrainte $c$ du problème (ainsi que la fonction objective) sous une forme linéaire du type $a_{c,1} * x_1 + a_{c,2} * x_2 + \ldots + a_{c,n} * x_n \leq b_c$, avec $x_1 \geq 0, \ldots, x_n \geq 0$ les variables du problème ($f_t^k$ par exemple dans notre modèle). Le modèle peut alors être écrit sous la forme :

$$\begin{cases} \text{minimiser } C^\top \mathrm{X} \\ AX \leq B \end{cases}$$

Avec $X = (x_0, \ldots, x_n)$, $C$ un vecteur de taille $n$ (nombre de variables), $B$ un vecteur de taille $m$ (nombre de contraintes) et $A$ une matrice de taille $m \times n$.

Afin de linéariser le modèle, plusieurs variables et contraintes doivent être ajoutées, et la fonction objective doit être reformulée.

### 3.5.1 Variables

Afin de linéariser la contrainte impliquant un $max$ (3.2), la quantité réelle de bande passante utilisée pour une session $k \in K$ sur un lien physique $(i,j) \in E$, $f^k(i,j)$, est maintenant définie par un ensemble de contraintes (cf. 3.14).

On utilisera pour cela deux nouveaux ensembles de variables, $g_t^k(i,j) \in \mathbb{B}$ et $g^k(i,j) \in \mathbb{B}$, indiquant la présence d'un flux sur le lien $(i,j)$ pour la session $k$, à destination de $t$ pour les premières. On ajoute la relation suivante entre $g_{(t)}^k(i,j)$ et $f_{(t)}^k(i,j)$ (cf. équation 3.15 pour la contrainte linéaire) :

$$\forall k \in K, \forall (i,j) \in E : g_t^k(i,j) = 0 \Leftrightarrow f_t^k(i,j) = 0 \tag{3.12}$$

$$\forall k \in K, \forall (i,j) \in E : g^k(i,j) = 0 \Leftrightarrow f^k(i,j) = 0 \tag{3.13}$$

On définit également deux variables $f_{max}$ et $c_{max}$ correspondant respectivement à la charge maximale des liens et des nœuds du réseaux physiques, pour l'ensemble des sessions de K, et qui seront utilisées dans les fonctions objectives (toujours afin de linéariser les $max$).

### 3.5.2 Contraintes

#### 3.5.2.1 Flux et bande passante

Pour linéariser le $max$ défini en 3.2, on ajoute la contrainte suivante :

$$\forall k \in K, \forall e \in E, \forall t \in T_k : f_t^k(e) \leq f^k(e) \tag{3.14}$$

Afin de satisfaire la contrainte définie en 3.12, on ajoute la contrainte linéaire suivante :

$$\forall k \in K, \forall e \in E : g^k(e) \leq f^k(e) \; et \; f^k(e) \leq b_k * g^k(e) \tag{3.15}$$

$$\forall k \in K, \forall t \in T_k, \forall e \in E : g_t^k(e) \leq f_t^k(e) \; et \; f_t^k(e) \leq b_k * g_t^k(e) \tag{3.16}$$

Afin de prendre en compte la contrainte définie en 3.7, il suffit de contraindre la partie gauche de l'équation 3.16, qui devient alors :

$$\forall k \in K, \forall e \in E : b_k^{min} * g_t^k(e) \leq f_t^k(e) \; et \; f_t^k(e) \leq b_k * g_t^k(e) \tag{3.17}$$

#### 3.5.2.2 Délai

On ajoute les contraintes suivantes concernant le délai de transmission des messages appartenant à une session $k$ :

$$\forall k \in K, \forall t \in T_k : d_t^k(s_k) = 0 \tag{3.18}$$

$$\forall k \in K, \forall t \in T_k, \forall (i,j) \in E : d_t^k(i) + \frac{p_k}{b_k^{min}} + d_{ij} - (1 - g_t^k(i,j)) * \Psi \leq d_t^k(j) \tag{3.19}$$

Avec $\Psi$ un nombre assez grand pour qu'un nœud sur lequel ne transite aucune donnée à destination de $t$ pour la session $k$ ne soit pris en compte dans l'équation précédente.

On notera ici que $\forall (i,j) \in E,\ b_k^{min} \leq f_t^k(i,j)$, et donc $\frac{p_k}{f_t^k(i,j)} < \frac{p_k}{b_k^{min}}$. Le délai obtenu après linéarisation est donc une majoration du délai obtenu par la modélisation normale (lui même une majoration du délai réel), la contrainte de délai réel est donc toujours respectée.

$$\forall k \in K, \forall t \in T_k : d_t^k(t) \leq d_k \tag{3.20}$$

La contrainte 3.20 est juste une copie de 3.8, alors que 3.18 et 3.19 remplacent la contrainte 3.3.

#### 3.5.2.3 Charge des nœuds

La contrainte 3.4 est remplacée en utilisant les $g^k(i,j)$ par 3.21.

$$\forall k \in K, \forall i \in V : c^k(i) = \sum_{\substack{j \in V \\ (j,i) \in E}} g^k(j,i) \tag{3.21}$$

#### 3.5.2.4 Charges maximales

Toujours pour linéariser les *max* utilisés dans la fonction objective 3.10, on ajoute les deux contraintes suivantes :

$$\forall e \in E : \frac{1}{c_e} * \left( \overline{c_e} + \sum_{k \in K} f^k(e) \right) \leq f_{max} \tag{3.22}$$

$$\forall i \in V : \frac{1}{u_i} * \left( \overline{u_i} + \sum_{k \in K} c^k(i) \right) \leq c_{max} \tag{3.23}$$

### 3.5.3 Fonction(s) objective(s)

La fonction objective définie en 3.10 devient simplement :

$$\text{minimiser } \alpha * f_{max} + \beta * c_{max} \tag{3.24}$$

La fonction objective définie 3.9 est quadratique. Bien qu'elle puisse être implémentée directement dans la plupart des solveurs, elle augmente de façon exponentielle le temps de résolution du problème.

À la place de cette fonction quadratique, la fonction suivante est utilisée :

$$\text{minimiser } \alpha_1 * f_{max} + \alpha_2 * S_E + \beta_1 * c_{max} + \beta_2 * S_V + \gamma * S_D \tag{3.25}$$

Où $S_E$, $S_V$ et $S_D$ représentent respectivement la somme des flux alloués, des capacités utilisés, des délais aux cibles et sont définis par :

$$S_E = \frac{\sum_{e \in E} \left( \frac{1}{c_e} * \left( \overline{c_e} + \sum_{k \in K} f^k(e) \right) \right)}{|E|} \tag{3.26}$$

$$S_V = \frac{\sum_{i \in V} \left( \frac{1}{u_i} * \left( \overline{u_i} + \sum_{k \in K} c^k(i) \right) \right)}{|V|} \tag{3.27}$$

$$S_D = \sum_{k \in K} \sum_{t \in T_k} d_t^k(t) \tag{3.28}$$

La fonction objective définie en 3.11 devient quant à elle :

$$\text{minimiser } \sum_{i \in V} c_b^k(i) \tag{3.29}$$

Avec $c_b^k(i) \in \mathbb{B}$, $c_b^k(i) = (c^k(i) + \overline{u_i} > 0)$ (cf. 3.15 pour la linéarisation de ce genre de contrainte).

### 3.5.4 Complexité

La complexité du modèle linéaire se retrouve dans le nombre de variables et contraintes nécessaires à sa résolution. Pour un réseau composé de $N$ routeurs et $M$ liens, l'allocation d'une requête composée de $K$ sessions, avec un maximum de $T$ destinations par session nécessite :

— $O(KT * (2M + N))$ variables.
— $O(KT * (4M + N))$ contraintes.

## 3.6 Conclusion

Nous avons présenté dans ce chapitre l'algorithme d'allocation de ressources que nous proposons pour le provisionnement des rséaux virtuels applicatifs sur une infrastructure SDN/Openflow. Cet algorithme a été implémenté et, également, évalué sur un réseau d'opérateur réel, à savoir le réseau Geant2 [3]. Ces évaluations n'ont pas été présentées dans ce livrable. Elles sont disponibles dans [8].

Les perspectives du travail de ce chapitre portent sur l'amélioration de l'algorithme et notamment (1) une prise en compte plus efficace d'un changement des caractéristiques du réseau virtuel et (2) une prise en compte plus précise des coûts des ressources de commutation.

# Chapitre 4

# Algorithme de déploiement de réseau virtuel

## 4.1 Introduction

L'objectif de l'algorithme présenté dans ce chapitre est de déployer sur une infrastructure SDN/Openflow un réseau virtuel constitué d'un ou plusieurs liens virtuels, chacun pouvant être de type point-à-point ou point-multipoint et chacun est caractérisé par un débit garanti. Il prend en entrée les allocations de ressources calculées par le module *Allocateur de ressource* et produit en sortie les messages Openflow [1] destinés à chaque commutateur de l'infrastructure. Ces derniers sont soumis au contrôleur Openflow qui prend en charge leur transmission jusqu'aux commutateurs. Cet algorithme repose sur un ensemble de pré-requis sur les éléments d'un commutateur Openflow ainsi que sur le protocole Openflow. Ils sont, dans un premier temps, rappelés dans la section suivante. La section 4.3 décrit l'algorithme proposé.

## 4.2 Quelques rappels et compléments sur Openflow

Un commutateur Openflow est constitué d'une ou plusieurs tables de flux ( *flow tables*), une table de groupes (*group table*) et une table de meters (*meter table*). Le premier type de table est utilisé pour exécuter des actions d'acheminement basiques et de modification des paquets, le second pour des actions d'acheminement plus complexes (diffusion, partage de charge, etc.) et le troisième pour exécuter des actions de régulation ou de mise en forme de trafic. Le contrôleur peut alimenter, mettre à jour et enlever des entrées de chacune de ces tables d'un switch en lui transmettant les messages du protocole Openflow suivants :

— Openflow Flow Modification Message (noté *ofp_flow_mod*) qui est utilisé par le contrôleur pour installer, enlever et mettre à jour des entrées de la table de flux d'un switch. Chaque entrée est composée d'une règle de correspondance ( *"match rule"*) qui identifie les paquets concernés par cette entrée et un ensemble d'instructions (appelé *instruction-set*) qui sont exécutées lors d'une correspondance d'un paquet. Ces instructions permettent de modifier le contenu du paquet et d'alimenter l'ensemble des actions cumulées (appelé *action-set*) qui seront ap-

1. comme indiqué ci-avant, le composant qui exécute cet algorithme s'interface avec une API nord de bas niveau

pliquées au paquet lorsqu'il sera commuté vers l'interface de sortie. Plusieurs instructions sont définies dans le protocole Openflow dont :
  - *write-actions* qui permet de rajouter des actions dans l'ensemble *action-set* du paquet. Parmi les actions figurent les moyens de modifier certains champs des paquets ou empiler/dépiler des champs, la redirection du paquet vers une interface de sortie ou vers un groupe de la *group table*.
  - *meter* qui permet d'aiguiller un paquet vers une entrée de la *meter table* pour des fins de régulation de tarfic
- Openflow Group Modification Message (noté *ofp_ group_ mod*) qui est utilisé par le contrôleur pour modifier le contenu de la *group table*. Une entrée de cette table contient un identificateur *group id* qui est utilisé par une entrée de la table de flux pour la désigner ainsi qu'une liste de *action buckets* qui permettent d'entrevoir différentes manières d'acheminer un paquet. Plusieurs types de groupes ont été définis par Openflow dont :
  - **Select** dont l'objectif est de permettre le partage de charge entre plusieurs chemins. A chaque bucket est associé un poids et une liste d'actions incluant une interface de sortie. Lorsqu'un paquet est dirigé vers un groupe de ce type, sur la base des poids de chaque bucket et du trafic déjà relayé par le groupe, un bucket est choisi pour prendre en charge l'acheminement du paquet.
  - **All** qui est utilisé pour réaliser du multicast ou du broadcast. Lorsqu'un paquet est dirigé vers un groupe de ce type, une copie de ce paquet est livrée à chaque bucket pour traitement.
- Openflow Meter Modification Message (noté *ofp_ meter_ mod*) qui permet au contrôleur de modifier le contenu de la *meter table*. Une entrée de cette table possède une identificateur (noté *meter-id*) et définit un régulateur de trafic potentiellement constitué de plusieurs régulateurs basiques appelés *meter bands*. A un *meter band* est associé un débit cible qu'il va devoir appliquer. Dans le cas où les paquets dirigés vers ce meter ne respectent pas ce débit, selon le type du meter band (*drop* ou *dscp-remark*), les paquets non conformes sont éliminés ou simplement remarqués avec un niveau de priorité faible.

Les entrées de la table des flux (*flow tables*) peuvent faire référence à des entrées de la *group table* ou de la *meter table*. Il est par conséquent nécessaire que ces dernières soient installées dans leur table respective avant toute référence depuis la table des flux. Ainsi, un contrôleur SDN doit s'assurer que les messages Openflow de modification de la table des meters précédent ceux relatifs à la modification de la table des groupes qui, à leur tour, précédent les messages de modification des tables de flux.

## 4.3 Algorithme de déploiement

L'algorithme a pour objet de construire l'ensemble des messages de modification des tables Openflow que le contrôleur aura à transmettre aux différents commutateurs de l'infrastructure qui supportent le réseau virtuel demandé. Ainsi, pour chacun de ces commutateurs, trois listes de messages seront construites, à savoir la liste des messages *ofp_ meter_ mod*, la liste des messages *ofp_ group_ mod* et la liste des messages *ofp_ flow_ mod*, puis transmises vers le commutateur dans cet ordre. Le mécanisme de *bundle* introduit dans la version 1.4 d'Openflow est utilisé pour permettre le stockage et la pré-validation de ces messages au niveau de chaque commutateur avant une confirmation globale sur tous les commutateurs concernés par le réseau virtuel à déployer. Ce mécanisme permet de synchroniser les changements à appliquer sur plusieurs com-

mutateurs et en cas de problème sur un commutateur donné, invalide les demandes de modification sur les autres commutateurs.
L'algorithme de déploiement est présenté ci-après. Très grossièrement :

1. Pour tout commutateur $i$ concerné par le réseau virtuel [ligne 2] et pour tout lien virtuel $k$ de ce réseau virtuel [ligne 3] :
   - **[ligne 5 - ligne 8]** : si le commutateur $i$ est le noeud source du lien virtuel $k$, insérer une demande de création d'un meter dans la liste des messages meterModMessageList de $i$ pour s'assurer qu'il ne dépasse pas le débit qui lui est garanti
   - **[ligne 21- ligne 33]** : si du "*path-splitting*" intervient au niveau du noeud $i$, insérer une demande de création de groupe dans la liste des messages *groupModMessageList* avant de demander la création de l'entrée de la table des flux des paquets de $k$ (à insérer dans la liste de messages *flowModMessageList*) avec comme action à rajouter dans l'*action-set* la redirection des paquets vers le groupe nouvellement créé avec préalablement le passage par le meter suscité dans le cas où $i$ est le noeud source ;
   - **[ligne 11 - ligne 21]** : si du "*path-splitting*" n'intervient pas au niveau du noeud $i$, demander la création de l'entrée de la table des flux des paquets de $k$ avec comme principale instruction de rediriger le paquet vers son interface de sortie avec un éventuel passage préalable par le meter si $i$ est le noeud source de $k$ ;
2. **[ligne 37 - FIN]** : Les listes relatives à chaque noeud étant constituées, les bundles sont créés au niveau de chaque commutateur pour récupérer dans l'ordre les messages relatifs à la table des meters puis ceux de la table des groupes puis ceux des tables de flux.

## Virtual network Deployer (V, E, K, f, F, V')

**Input :**

V : is the set of nodes (Openflow Switches). Each node i procresses and forwards data to another node j or to rest of the network via a port noted $p_{ij}$

E : is the set of links. The link between the nodes i and j is noted (i,j).

K: is the set of virtual links. Each virtual link k is characterised by :

a source node $s_k$ element of V, a bandwith $b_k$, and a set of destination nodes $T'_k$ part of V except $s_k$

f : is a 4-dimensional dictionary of integer noted $f_k^t(i,j)$, representing bandwidth allocated at link (i,j) to packets of virtual link k, that are flowing from the source node $s_k$ to the destination note t.

F : is a 3-dimensional dictionary of integer noted $f_k(i,j)$, representing bandwidth allocated at link (i,j) to packets of virtual link k.

V': is the set of nodes crossed by at least one virtual link.

**Var :** match : ofp_match; group_mod : ofp_group_mod; nextHops a set of nodes; flowModMessageList : ofp_flow_mod[]; groupModMessageList : ofp_group_mod[]; meterModMessageList : ofp_meter_mod[]; groupID, meterID, bundleID : integer;

```
1   begin
2   for each i in V' do
3       for each unicast link k in K
4           // if the node i is the source node of virtual link k, then add a meter Mod in meterModMessageList and keep
5   meterID to associate with flowMod as instruction        if(i = s_k) then
6               meterID ← get_meter_id(i,k)
7               insert_into (meterModMessageList, {id = meterID, bands [0] = {rate = b_k, type = drop}})
8           end if
9           nextHops ←  computeNextHops(i, k,V', E, f)
10          // if the virtual link k isn't split at node i, add a flow Mod in flowModMessageList with output action; and
11      meter instruction if i is the source of virtual link k
12          if (|nextHops| = 1) then
13                  for each j  in nextHops do
14              if(i = s_k) then
15                  insert_into (flowModMessageList, { match = create_match(), instructions = {meter: meterID, write-
16              actions: output.port = Pij})
17              else
18                  insert_into (flowModMessageList, { match = create_match(), instructions = {write-actions:
19              output.port = Pij})
20              end if
21          end for
22          // if the virtual link k is split at node i, add a group Mod in groupModMessageList and add a flow Mod in
23          flowModMessageList with that group as action; and meter instruction if i is the source of virtual link k
24          else
25              groupID ← get_group_id(i,k)
26              group_mod.id ← groupID; group_mod.type ← select
27              for each j in nextHops do
28                  group_mod.buckets[j] ← {weight = f_k(i,j), actions = {ouput.port = Pij} }
```

```
            end for
            insert_into (groupModList, group_mod)
            if (i = s_k) then
                insert_into (flowModMessageList, { match = create_match(), instructions = {meter: meterID, write-
    actions: group.id = groupID}})
            else
                insert_into (flowModMessageList, match = create_match(), instructions = {write-actions: group.id =
    groupID}})
            end if
        end if
    end for
    // at this step, meterModMessageList, groupModMessageList, flowModMessageList contain all messages to
    convey at the node i, for installing VNET. This last blocks use bundle with ordered flag set, to request the
    message of the bundle are applied strictly in order.
    bundleID ← get_bundle_id(i)
    OFPBCT_OPEN_REQUEST {id = bundleID, flags = ordered}
    for each msg in meterModMessageList do
        OFPT_BUNDLE_ADD_MESSAGE {id = bundleID, message = msg}
    end for
    for each msg in groupModMessageList do
        OFPT_BUNDLE_ADD_MESSAGE {id = bundleID, message = msg}
    end for
    for each message in flowModMessageList do
        OFPT_BUNDLE_ADD_MESSAGE {id = bundleID, message = msg}
    end for
    reset (meterModMessageList) ; reset (groupModMessageList), reset (flowModMessageList)
end for
for each i in V' do
    OFPBCT_CLOSE_REQUEST {id = get_bundle_id(i)}
    OFPBCT_COMMIT_REQUEST {id = get_bundle_id(i)}
end for

end
.....
nextHops computeNextHops(i, k,V', E, f )
Begin
nextHops ← ∅
for each j in V'  do
    if( (i, j) in E and f_k^t (i,j)>0) then
            add j in nextHops
    end if
end for
return nextHops

end
```

# Conclusion

Ce document est le deuxième livrable de la tâche 1 (système de communication – Architectures, algorithmes et mécanismes) du projet ADN dont l'objectif est de présenter, à mi-parcours, l'architecture préliminaire ADN ainsi que les algorithmes des composants de la fonction réseau, au coeur de cette architecture, en charge de provisionner et de maintenir un réseau virtuel applicatif pour le compte d'applications dynamiques DDS. Les algorithmes des composants en charge de la *capture des besoins applicatifs et de leur retraduction en une demande de service réseau*, de l'*allocation de ressources* et du *déploiement du réseau virtuel* ont été détaillés. Les exigences préliminaires des composants de *monitoring réseau* et de *gestion autonomique* ont été identifiées.

Un bilan objectif sur l'avancement des travaux du projet est décliné ci-après. La démarche suivie dans ce projet pour développer le concept de réseau ADN repose sur une connaissance très fine des besoins applicatifs, explicitement communiqués au réseau. Ce choix se justifie par la volonté d'offrir des services réseau qui répondent à des exigences précises de l'application tout en utilisant au mieux les ressources réseau. C'est clairement la spécificité de notre approche. En effet, les approches ADN de la littérature reposant sur une description explicite des besoins applicatifs adoptent un niveau de granularité très large avec, au final, une connaissance beaucoup moins précise du trafic applicatif et de ses besoins mais, en revanche, une meilleure résistance à l'échelle.
Les grandes lignes de l'architecture ADN ont été présentées dans ce rapport. La décomposition fonctionnelle proposée pour le contexte filaire est suffisamment générale pour rester valable dans un contexte sans-fil. Un travail supplémentaire est nécessaire pour expliciter exhaustivement les exigences des composants de monitoring réseau et de gestion autonmique. Ce travail est en cours.

L'algorithme d'allocation de ressources que nous avons proposé présente plusieurs spécificités par rapport aux travaux de la littérature, dont : la prise en compte (1) des liens point-multipoints, (2) de plusieurs critères de QoS et (3) des ressources de commutation dans le processus d'allocation. Néanmoins, la réallocation de ressources suite à une changement des besoins applicatifs ne prend pas en compte la possibilité de permettre une migration de ressources ce qui conduirait à des solutions proches de l'optimum avec un coût réduit. Ce point fera l'objet de nos actions à venir ainsi que d'autres actions qui visent à prendre en compte de manière encore plus précise le coût d'une ressource de commutation.

L'algorithme de déploiement proposé s'applique à un réseau virtuel qui combine des liens virtuels point-à-point (avec la possibilité d'activer le *path splitting*) avec des liens point-multipoints (sans activation du *path splitting*). L'algorithme permettant le *path splitting* pour des liens virtuels point-multi-points est en cours d'élaboration.

Dans la perspective du projet ADN, les résultats décrits dans ce rapport sont les points d'entrée des développements, en cours de finalisation, du premier prototype du

projet qui concerne l'application de l'approche SDN à une application de simulation distribuée s'exécutant dans un environnement maîtrisé. Ces travaux font l'objet de la tâche 2 du projet.

# Bibliographie


[1] N.M.M.K. Chowdhury, M.R. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*, pages 783–791, April 2009. doi : 10.1109/INFCOM.2009.5061987.

[2] N.M.M.K. Chowdhury, M.R. Rahman, and R. Boutaba. Vineyard : Virtual network embedding algorithms with coordinated node and link mapping. *Networking, IEEE/ACM Transactions on*, 20(1) :206–219, Feb 2012. ISSN 1063-6692. doi : 10.1109/TNET.2011.2159308.

[3] DANTE. GÉANT, Media Library : Maps, 2014. URL `http://www.geant.net/Resources/Media_Library/Pages/Maps.aspx`.

[4] Open Networking Foundation. Openflow notification framework. Technical report, [Online]. Available : https ://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-notifications-framework-1.0.pdf, 2013.

[5] Open Networking Foundation. Openflow configuration and management protocol (of-config) 1.2. Technical report, [Online]. Available : https ://www.opennetworking.org/sdn-resources/onf, 2014.

[6] Christian Frei and Boi Faltings. Resource allocation in networks using abstraction and constraint satisfaction techniques. In Joxan Jaffar, editor, *Principles and Practice of Constraint Programming – CP'99*, volume 1713 of *Lecture Notes in Computer Science*, pages 204–218. Springer Berlin Heidelberg, 1999. ISBN 978-3-540-66626-4. doi : 10.1007/978-3-540-48085-3_15. URL `http://dx.doi.org/10.1007/978-3-540-48085-3_15`.

[7] Wu-Hsiao Hsu, Yuh-Pyng Shieh, Chia-Hui Wang, and Sheng-Cheng Yeh. Virtual network mapping through path splitting and migration. In *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, pages 1095–1100, March 2012. doi : 10.1109/WAINA.2012.74.

[8] M. Capelle S. Abdellatif M.J. Huguet and P. Berthou. Online virtual links resource allocation in software-defined networks. In *IFIP Networking*, 2015.

[9] M. Kucharzak and K. Walkowiak. Maximum flow trees in overlay multicast : Modeling and optimization. In *Future Internet Communications (BCFIC), 2012 2nd Baltic Congress on*, pages 260–267, April 2012. doi : 10.1109/BCFIC.2012.6217955.

[10] Jens Lischka and Holger Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *Proceedings of the 1st ACM Workshop on Virtualized Infrastructure Systems and Architectures*, VISA '09, pages 81–88, New

York, NY, USA, 2009. ACM. ISBN 978-1-60558-595-6. doi : 10.1145/1592648.1592662. URL http://doi.acm.org/10.1145/1592648.1592662.

[11] ONF Market Education Committee. Software-defined networking : The new norm for networks. *Open Network Foundation*, 2012.

[12] H. Masri, S. Krichen, and A. Guitouni. An ant colony optimization metaheuristic for solving bi-objective multi-sources multicommodity communication flow problem. In *Wireless and Mobile Networking Conference (WMNC), 2011 4th Joint IFIP*, pages 1–8, Oct 2011. doi : 10.1109/WMNC.2011.6097256.

[13] M. Melo, J. Carapinha, and S. Sargento. Network virtualization : A step closer for seamless resource mobility. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 692–695, May 2013.

[14] M. Melo, S. Sargento, U. Killat, A. Timm-Giel, and J. Carapinha. Optimal virtual network embedding : Node-link formulation. *Network and Service Management, IEEE Transactions on*, 10(4) :356–368, December 2013. ISSN 1932-4537. doi : 10.1109/TNSM.2013.092813.130397.

[15] J. Nogueira, M. Melo, J. Carapinha, and S. Sargento. Virtual network mapping into heterogeneous substrate networks. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 438–444, June 2011. doi : 10.1109/ISCC.2011.5983876.

[16] B. Nunes, Marc Mendonca, Xuan-Nam Nguyen, K. Obraczka, and Thierry Turletti. A survey of software-defined networking : Past, present, and future of programmable networks. *IEEE Communications Surveys and Tutorials*, 2014.

[17] OMG. Data distribution service for real time systems specification, ddsv1.2. In *http ://www.omg.org/spec/DDS/1.2/*, 2007.

[18] OMG. Data distribution service interoperability wire protocol specification. ddsi v2.1. In *http ://www.omg.org/spec/DDSI/2.1/*, 2009.

[19] A.Hakiri P.Berthou and S.Abdellatif. Publish/subscribe-enabled software defined networking for efficient and scalable iot communication. *to appear in IEEE Communication Magazine*, 2015.

[20] L. Bertaux A.Hakiri S.Medjiah P.Berthou and S.Abdellatif. A dds/sdn based communication system for efficient support of dynamic distributed real-time applications. In *IEEE/ACM DS-RT*, 2014.

[21] M.S. Rathore, M. Hidell, and P. Sjodin. Performance evaluation of open virtual routers. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 288–293, Dec 2010. doi : 10.1109/GLOCOMW.2010.5700328.

[22] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker. Modular sdn programming with pyretic. In *USENIX*, 2013.

[23] S. Abdellatif L.Bertaux P. Berthou S. Medjiah A. Hakiri T. Villemur T. Gayraud M.Diaz T.Monteil. Livrable 1.1 : Etat de l'art : Sdn, virtualisation réseau et dds. Technical report, Rapport de contrat du projet ANR/DGA ADN, oct. 2014.

[24] F.Simo-Tegue P.Berthou S.Abdellatif T.Villemur and F.Mkacher. Dds et sdn en symbiose pour les applications dynamiques. In *CFIP*, 2015.

[25] A. Voellmy, H. Kim, and N. Feamster. Procera : a language for highlevel reactive network control. In *first workshop on Hot topics in software defined networks (HotSDN '12)*, 2012.

[26] Yongtao Wei, Jinkuan Wang, Cuirong Wang, and Xi Hu. Bandwidth allocation in virtual network based on traffic prediction. In *Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on*, pages 1–4, Sept 2010. doi : 10.1109/WICOM.2010.5601426.

[27] Yufang Xi and E.M. Yeh. Distributed algorithms for minimum cost multicast with network coding. *Networking, IEEE/ACM Transactions on*, 18(2) :379–392, April 2010. ISSN 1063-6692. doi : 10.1109/TNET.2009.2026275.

[28] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding : Substrate support for path splitting and migration. *SIGCOMM Comput. Commun. Rev.*, 38(2) :17–29, March 2008. ISSN 0146-4833. doi : 10.1145/1355734.1355737. URL `http://doi.acm.org/10.1145/1355734.1355737`.