Architecture finale et algorithmes pour le cas d'applications en environnement dynamique

Armel Francklin Simo Tegueu, Pascal Berthou, Slim Abdellatif, Thierry Villemur

Livrable 1.3 Architecture finale et algorithmes pour le cas d'applications en environnement dynamique

Project name	ADN
Project ID	ASTRID xxx
Working Package Number	Tache x
Deliverable Number	numéro x
Document title	Template for Report
Document version	0.1
Editor in Chief	Last Name, Affiliation
Authors	
Date	01/02/2017
Reviewer	Last Name, Affiliation
Date of Review	15/12/2016
Status	*Public*

Revision History

Date	Version	Description	Author
01/07/15			
02/07/15			
03/07/15			
04/07/15			

Table des matières

Résumé

Introduction

Chapitre 1

Partie1 : Architecture ADN finale

1.1 Architecture finale dans le cas d'une environnement maîtrisé

Décrire rapidement l'archi et expliciter l'incrément par rapport à l'archi préliminaire du livrable 1.2.
Il faudrait mettre les diagrammes UML de l'archi.

1.2 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 [**?**], 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 applicatifs et, de l'autre côté, avec le contrôleur SDN pour déployer les service associés.

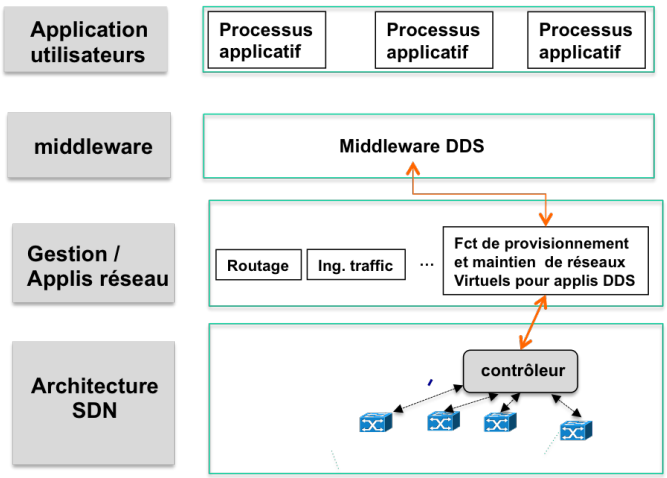

FIGURE 1.1 – Architecture générale

La figure 1.2 présente de manière générale les différentes alternatives qui

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

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 [?] ou Procera [?], 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).

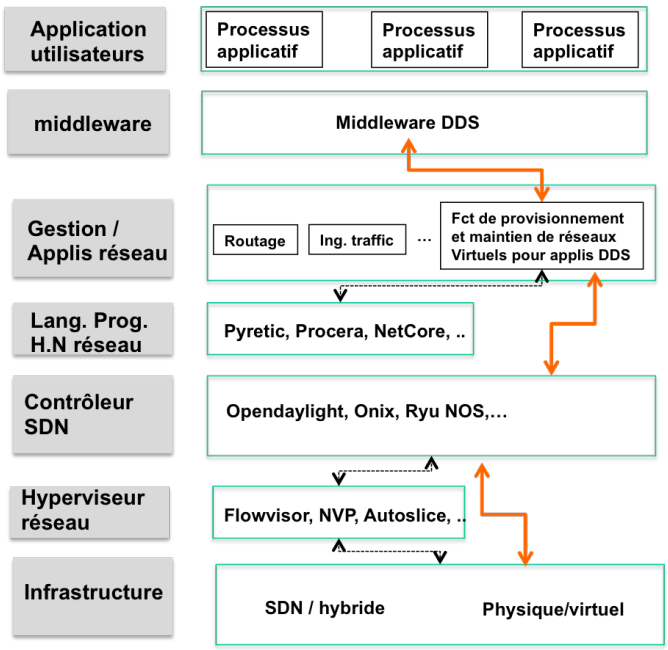

FIGURE 1.2 – Choix de conception relatifs à la fonction de provisionnement

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)[**? ?**] :

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

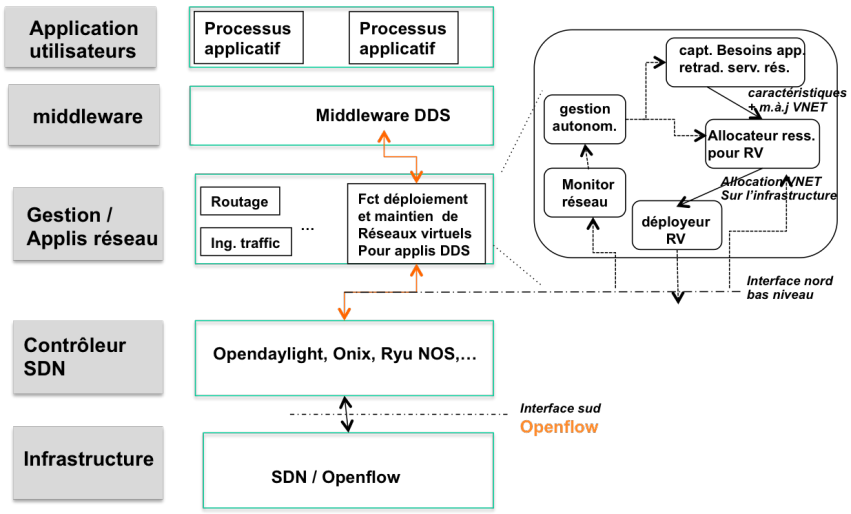

FIGURE 1.3 – Composants de la fonction de provisionnement

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 auto-*

nomique" 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.

1.3 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/désouscription 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*)[**?**]. 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.

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

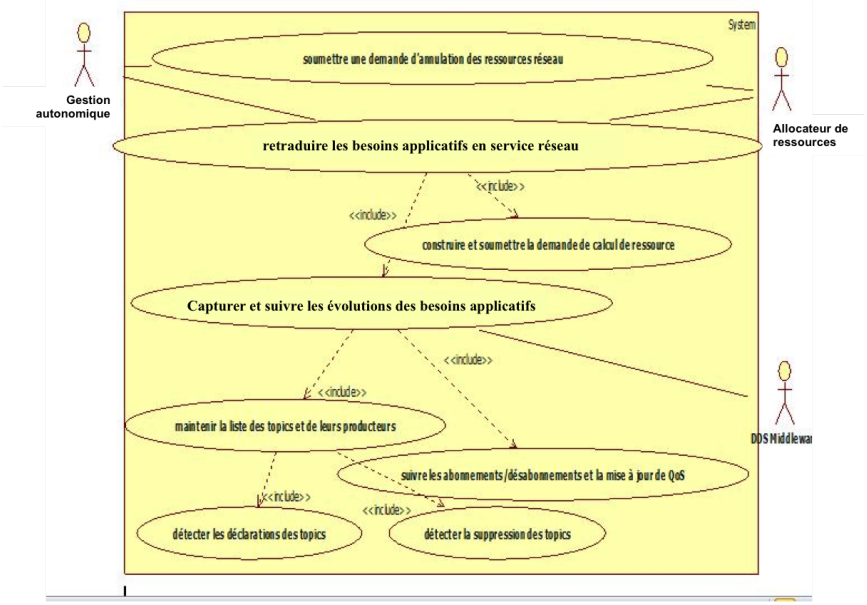

FIGURE 1.4 – Diagramme des cas d'utilisation du composant "capture de besoins et retraduction en service réseau"

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.

1.4 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.

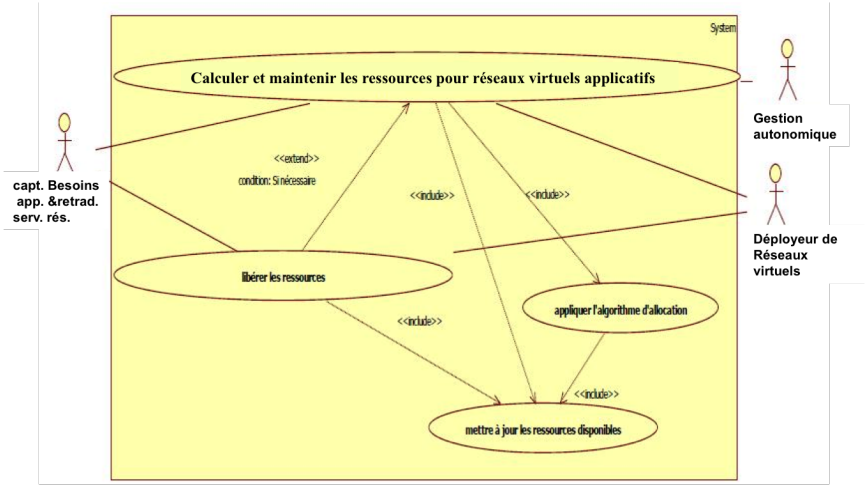

FIGURE 1.5 – Diagramme des cas d'utilisation du composant allocateur de ressources

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.

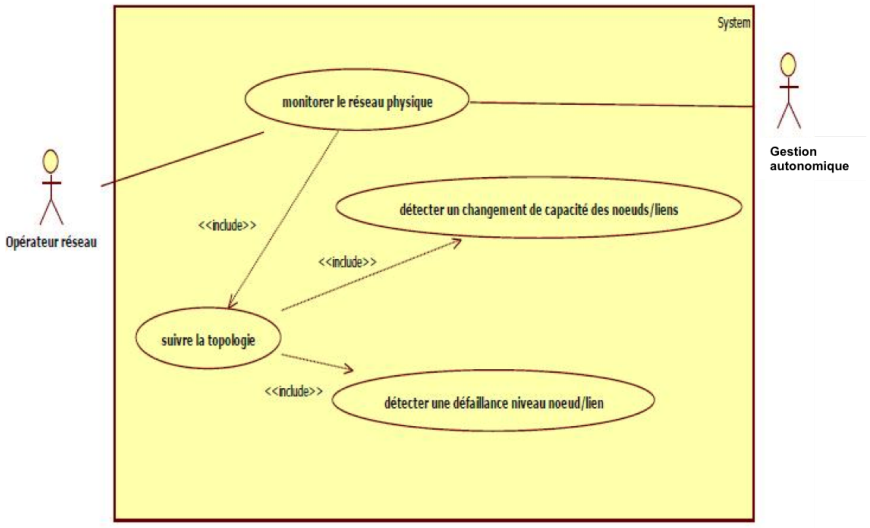

FIGURE 1.6 – Diagramme des cas d'utilisation du composant Monitor réseau

1.5 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 [**?**], énumère une liste d'évènements, en plus de ceux déjà évoqués dans les spécifications OF-Config [**?**] 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.

1.6 Déployeur de réseau virtuel

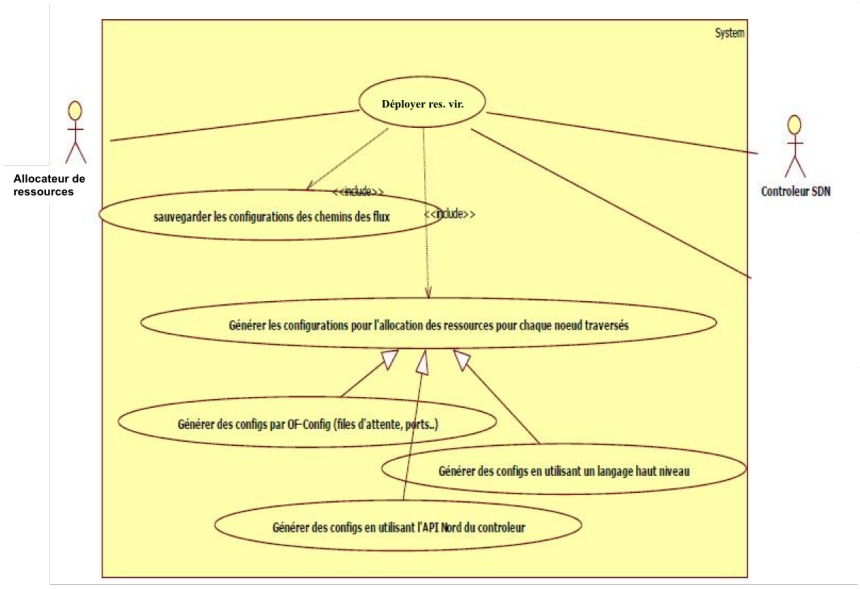

FIGURE 1.7 – Cas d'utilisation du composant Déployeur réseau virtuel

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 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.

1.7 Modifications/Extensions dans le cas d'une environnement dynamique

1.8 Positionnement par rapport à l'existante

Décrire les spécificités, forces, faiblesses de ADN.

1.9 Perpectives

Chapitre 2

Partie 2 : Algorithmes dans le cas maîtrisé

2.1 Algorithme d'allocation finale (dynamique, ..)

Travail en cours de Francklin

2.2 Algorithme de déploiement

présentation trè sapide et renvoi vers [ref. Livrable 1.2]

2.3 Algorithme Application Requirements to NET mapper

2.3.1 application related traffic Estimation (Elementary flow estimation)

cas 1 : besoins explicitement exprimés par l'application

Cas 2 : Besoins non exprimés

2.3.2 Algorithme d'agrégation de flux

Objectifs de préciser les algorithmes des composants "application requirements to VNET mapper" et l'algorithme de déploiement. Point de départ sont les algorithmes développés dans le cas maîtrisé pour voir s'il est nécessaire de les mettre à jour. Le produit de cette tâche alimente la partie 3 du livrable L1.3.

2.3.3 Algorithme du manager autonomique

Objectif est de décrire l'algorithme de l'autonomic manager avec le framework frameself. Il faudra présenter le framework frame-self et pour les objectifs identifiés dans la tâche T1.1, décliner le fonctionnement de l'autonomic manager (règles dans chaque phase).

Chapitre 3

Partie 3 : Algorithmes cas dynamique

3.1 Analyse de l'applicabilité de l'approche SDN aux réseaux multi-sauts sans-fil

Objectif est : - d'identifier les principaux défis relatifs à l'application de l'approche SDN aux réseaux multi-sauts sans-fil et travaux existants (travail d'état de l'art) : pb de délais, pertes messages Openflow, etc. - Résultats expérimentaux sur cetta applicabilité dans le cas d'une signalisation Openflow dans la bande our hors bande basée sur des réseaux sans fil à one-hop du style LoRA ou SATCOM ou cellulaire ?

3.2 Algorithme du network discovery

3.3 Algorithme d'allocation de ressources

Objectif est toujours de trouver une route et réserver les ressources le long de cette route. Point de départ, voir comment adapter la formulation précédent au cas sans-fil + voir s'il y a pas d'autres ressources à prendre en compte dans la modélisation. Les résultats alimenteront la partie 3 du livrable 1.3

3.4 Algorithme du manager autonomique

3.5 Autres algorithmes : Estimation de traffic, aggro flux, etc.

Conclusions

3.6 Acronymes