TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

SDN	Software Defined Network
NFV	Network Function Virtualization
NF	Network Function
VNF	Virtualized Network Function
NFV MANO	NFV management and network orchestration
SFC	Service Function Chain
IDS	Intrusion Detection System
NAT	Network Address Translation
API	Application Programming Interface
NIST	National Institute of Standards and Technology
ONF	Open Networking Foundation
ETSI	European Telecommunications Standards Institute
OPEX	Operational Expenditure
CAPEX	Capital Expenditure
ILP	Intger Linear Programming
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service

INTRODUCTION

Contexte

Jusqu'à récemment, l'Internet a réussi à offrir et gérer divers services et applications distribuées avec succès. Cependant, l'immense popularité d'Internet s'est également révélée être son plus grand obstacle à l'évolution et l'innovation. En effet, avec l'apparition des services infonuagiques et la diversité des applications réseau, la quantité des données augmente exponentiellement. Toutefois, la rigidité de l'architecture Internet l'empêche à suivre cette croissance menant à ce qu'appelle le problème d'ossification. De plus, en raison de sa nature multifournisseur, adopter une nouvelle architecture ou modifier celle qui existe est une tâche difficile et requiert un consensus entre des multiples parties prenantes. En conséquence, les modifications apportées à l'Internet actuel se limitent à des correctifs incrémentiels, à savoir l'investissement dans des équipements suffisamment puissants pour gérer ces données, ce qui a entraîné une augmentation des dépenses d'exploitation (OPEX) et des dépenses en investissement (CAPEX).

L'émergence des technologies de virtualisation des fonctions réseau (Network Function Virtualization - NFV) et de la réseautique définie par logiciel (Software Defined Networks - SDN) est en train de transformer la conception et la gestion des réseaux informatiques. Ces technologies offrent aux opérateurs plus de flexibilité pour approvisionner des services réseau et configurer le réseau de manière dynamique. Cela permet de pallier les limitations des réseaux traditionnels.

La technologie NFV apporte une toute nouvelle dimension au paysage du marché de l'industrie des télécommunications grâce à la possibilité de réduire les investissements en capital, la consommation d'énergie en consolidant les fonctions de réseau et en introduisant des services personnalisés en fonction des besoins des clients. En effet, cette technologie révolutionne la manière de la conception des infrastructures en créant de manière dynamique des fonctions réseau sous la forme d'instances logicielles séparées de tout matériel dédié. Ces instances

logicielles appelées fonctions de réseau virtuelles (Virtual Network Functions - VNFs) sont instanciées et exécutées sur des serveurs commerciaux standards sans le besoin d'installer de nouveaux équipements.

En outre, NFV simplifie le déploiement de services en exploitant le concept de chaînage de services. Pour fournir un service Internet spécifique (par exemple, VoIP, Web Service, etc.), les fournisseurs conçoivent leurs services sous forme d'une concaténation des fonctions de réseau virtuelles (VNFs), telles que des routeurs, des IDS et des NAT, exécutés sur des machines virtuelles afin de traiter le trafic entrant et de le diriger vers sa destination. Cette chaîne de fonctions de réseau virtuelles est, ainsi, appelée une chaîne de service (Service Fuction Chain - SFC).

Au cours des dernières années, de nombreux travaux ont été consacrés au problème d'approvisionnement de ressources et à la gestion de ces SFCs, (Bari, Boutaba, Esteves, Granville, Podlesny, Rabbani, Zhang & Zhani (2013) ; Herrera & Botero (2016) ; Luizelli, Bays, Buriol, Barcellos & Gaspary (2015) ; Mijumbi *et al.* (2016) ; Qu, Assi & Shaban (2016) ; Racheg, Ghrada & Zhani (2017)). Ce problème consiste à trouver l'emplacement optimal des composants des chaînes de services satisfaisant les exigences en termes des ressources (CPU, mémoire et disque). La plupart des études existantes supposent une disponibilité totale de l'infrastructure physique, ce qui n'est pas réaliste, car les défaillances sont courantes dans les infrastructures de réseau infonuagique (ETSI (2015a) ; Harris (2018) ; Lee, Ko, Suh, Jang & Pack (2017) ; Zhang, Zhani, Jabri & Boutaba (2014b)). En raison de la dépendance entre les fonctions de réseau virtuelles dans la chaîne de service, une défaillance d'un seul nœud physique dans le réseau pourrait facilement mettre hors service de nombreux VNFs et, par conséquent, briser plusieurs SFCs et rendre les services, dont elles offrent, indisponibles. Ces temps d'interruption de services, même pendant quelques secondes, non seulement nuisent à la réputation des fournisseurs de

services, mais entraînent également des pertes de revenus élevées en fonction du type de service proposé (par exemple, $5600 par minute selon COHEN).

Problématique

Comme indiqué précédemment, pour construire une SFC, les VNFs sont connectés via un ensemble de liens virtuels ayant une quantité de bande passante suffisante pour transporter le trafic. Généralement, les chaînes de fonctions de service sont placés dans une infrastructure physique (appelée infrastructure NFV - NFVI) ETSI (2015a). La figure 0.1 montre un exemple de trois chaînes de services placées dans une infrastructure NFV de grande étendue. La figure montre, pour chaque chaîne, comment les VNFs sont placés de la source à la destination. Par exemple, la chaîne 2 a un trafic en provenance du nœud physique 2 vers le nœud physique 13 et elle est composée de trois VNFs de type 1, 2 et 3 qui sont inclus dans les nœuds physiques 3, 6 et 10, respectivement. La chaîne 1 ne comporte que deux VNFs de type 1 et 3 deployés sur des nœuds physiques 4 et 11, respectivement. Quant à la chaîne 3 qui est composée aussi de deux VNFs de type 1 et 2, NF1 est placé dans le nœud physique 7 alors que NF2 est placé dans le nœud physique 11.

Une fois que les SFCs sont placées dans l'infrastructure NFVI, l'opérateur doit assurer la disponibilité de ces SFCs. En d'autres termes, les SFCs doivent survivre aux potentielles défaillances du réseau afin de minimiser les interruptions de service. Cependant, les nœuds physiques sont enclins aux pannes et une défaillance d'un seul nœud peut entraîner l'échec de plusieurs VNFs et, par conséquent, la rupture de plusieurs chaînes de services.

Nous proposons, dans ce travail, d'assurer la survie des SFCs affectés par une seule défaillance en exploitant des sauvegardes partagées (shared backups) entre les VNFs de même type placés dans l'infrastructure et qui peuvent être utilisées en cas de défaillance. Nous supposons également qu'un VNF de sauvegarde doit être du même type que l'ensemble des VNFs qu'il sauvegarde. En

d'autres termes, un VNF de type i ne peut sauvegarder que des VNFs de type i. Cette hypothèse est raisonnable car, un VNF de sauvegarde doit contenir exactement la même pile de logiciels que le VNF d'origine afin d'éxécuter la même fonction en cas de panne.

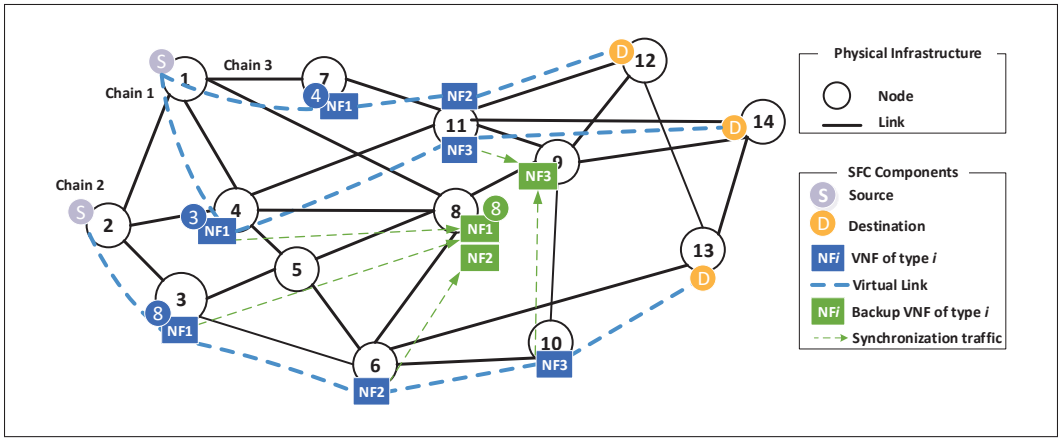


Figure 0.1 Un exemple d'approvisionnement des sauvegardes partagées.

Comme exemple de la manière dont les sauvegardes partagées peuvent être placées, nous pouvons voir sur la figure 0.1 que le nœud physique 9 héberge un VNF de sauvegarde de type 3 (c'est-à-dire, NF3) partagée entre le VNF de type 3 de la chaîne 1 et celui de la chaîne 2. Si le nœud physique 11 échoue et que NF3 de la chaîne 1 devient hors service, la sauvegarde NF3 hébergée dans le nœud physique 9 prend le relais et remplace la fonction défaillante. De même, il peut reprendre le service de NF3 appartenant à la chaîne 2 (hébergé dans le nœud 10) en cas d'échec. La figure montre également d'autres exemples de VNF de sauvegardes partagées (par exemple, NF1 et NF2 hébergés dans le nœud 8). On peut vérifier que les deux chaînes de services illustrées dans cet exemple peuvent parfaitement survivre à toute défaillance d'un seul nœud.

De plus, les VNFs de sauvegarde sont synchronisés en permanence avec les VNF actifs pour être prêts à prendre en charge le service en cas de panne (Flèches vertes sur la Figure 0.1). Par exemple, la sauvegarde NF3 hébergée dans 9 a l'état de NF3 hébergé dans le nœud physique 11 et celle de NF3 hébergée dans 10. Chaque fois qu'une défaillance survient, le dernier état

de la fonction défaillante sera utilisé lors de l'activation de la sauvegarde. La synchronisation d'état peut être effectuée à des niveaux différents : au niveau de la machine virtuelle exécutant la fonction (par exemple, synchronisation de la mémoire Boutaba, Zhang & Zhani (2013)) ou en utilisant des scripts de synchronisation personnalisés en fonction du type de la fonction réseau (par exemple, les règles de synchronisation dans les pare-feu).

Pour garantir la performance de la synchronisation d'état, la latence entre un VNF et sa sauvegarde ne doit pas dépasser une certaine limite. De plus, la synchronisation de l'état VNF peut consommer une bande passante. Dans notre travail, nous nous assurons de minimiser le délai de synchronisation et la bande passante consommée en limitant le nombre de sauts entre chaque VNF et sa sauvegarde (par exemple, le nombre de sauts est limité à 2 dans la Figure 0.1).

Nous proposons aussi de profiter de la migration des VNFs afin d'augmenter le nombre de VNFs doté de sauvegardes et de réduire le nombre total des sauvegardes partagées. Par exemple, la figure 0.1 montre que certains VNFs peuvent rester sans sauvegarde comme le cas de VNF NF1 de la chaîne 3. Nous proposons alors de rapprocher ces VNFs des sauvegardes approvisionnées pour en profiter.

La figure 0.2 montre comment le VNF NF1 de la chaîne 3 a été migré du nœud 7 vers le nœud 4.

La migration pourrait également être utilisée pour réduire le nombre de sauvegardes en distribuant les VNFs équitablement entre les nœuds proches de celui hébergeant les sauvegardes comme illustré dans la figure 0.3.

Le principal défi que nous abordons dans ce travail consiste à trouver le nombre minimal des VNFs de sauvegarde et à déterminer leur emplacement optimal dans l'infrastructure physique pour chaque type de VNF en tenant compte du délai de synchronisation.

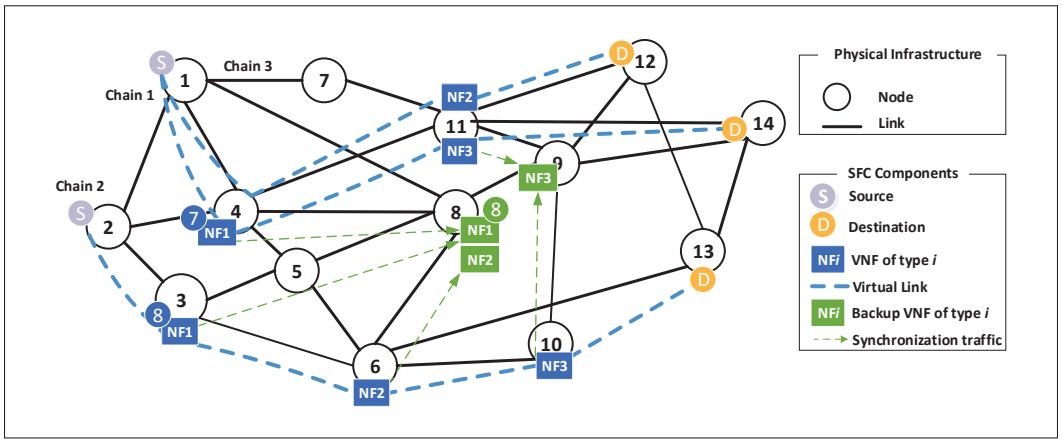


Figure 0.2 La migration des VNFs pour profiter des sauvegardes partagées.

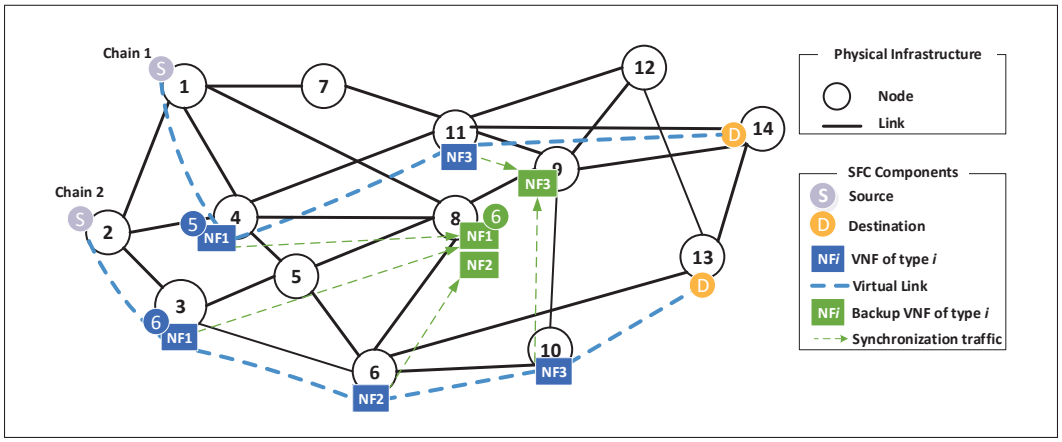


Figure 0.3 La migration des VNFs pour optmiser le nombre des sauvegardes partagées.

Objectif et contributions

Notre objectif principal est d'assurer la disponibilité de tous les SFCs intégrés dans l'infrastructure physique contre toute défaillance d'un seul nœud physique. Nous atteignons cet objectif en fournissant de manière proactive le nombre minimal des sauvegardes afin de minimiser le gaspillage de ressources et en les plaçant dans l'infrastructure. Nous prenons également en compte le coût de synchronisation en termes de bande passante et le délai nécessaire pour

maintenir les nœuds de sauvegarde à jour. Ainsi, nous proposons un système de gestion des ressources avec un module de disponibilité. Ce module permet de provisionner et de gérer les VNFs de sauvegarde. Il peut être facilement intégré aux infrastructures de gestion de ressources SFCs existantes. Nous formulons, ensuite, le problème d'approvisionnement et de placement de sauvegarde sous forme d'un programme linéaire en nombres entiers ILP (Integer Linear Program) qui permet de trouver le nombre optimal de sauvegardes partagées pour chaque type de VNF et de les placer de manière optimale dans l'infrastructure physique. Finalement, nous avons conçu cinq algorithmes heuristiques qui visent à résoudre le problème pour des scénarios à grande échelle dans un délai raisonnable. Ces algorithmes se basent sur la recherche des potentiels hôtes pour héberger les sauvegardes, mais se diffère dans la façon dont les hôtes sont finalement choisis.

Méthodologie du travail

Pour atteindre notre objectif et répondre aux questions de la problématique, la méthodologie que nous proposons est subdivisée en trois parties :

- nous avons commencé par une exploration du contexte de notre recherche en mettant l'accent sur les concepts relatifs à l'infonuage et la virtualisation des fonctions du réseau. Nous analysons, aussi, les solutions antérieures qui abordent notre problématique dans le but d'établir l'état de l'art pour le problème de disponibilité des ressources dans l'infonuage ;
- nous avons passé à la conception du système, modélisé mathématiquement le problème et proposé des solutions qui permettront de trouver le nombre minimal des nœuds de sauvegardes et leurs emplacements ;
- finalement, nous avons évalué la performance des heuristiques proposées et les comparé à la solution optimale trouvée avec l'ILP proposé résolu par CPLEX.

Publications

Durant la période du ce projet de maîtrise, nous avons réussi à publier deux articles.

Nous avons publié un article intitulé «On Improving Service Chains Survivability Through Efficient Backup Provisioning» (Aidi, Saifeddine et Zhani, Mohamed Faten, 2018) qui a été présenté à la conférence «15th International Conference on Network and Service Management CNSM2018» à Rome (Italie) en novembre 2018 où nous avons proposer deux algorithmes visant à approvisionner un nombre minimal de sauvegardes pour assurer la survie des SFCs en cas d'une panne.

Par la suite, nous avons publié un deuxième article intitulé «On Optimizing Backup Sharing Through Efficient VNF Migration» qui a été présenté au Workshop «IEEE PVE-SDN 2019» qui a eu lieu avec «5th IEEE International Conference on Network Softwarization NetSoft2019» à Paris (France) en juin 2019, où nous abordons certaines limites de la solution proposée dans le premier article en proposant deux algorithmes basés sur la migration.

Plan du rapport

Le reste de ce rapport est organisé comme suit. Le premier chapitre présente une introduction générale sur les concepts de l'infonuagique et la virtualisation.

Le deuxième chapitre, nous présentons une revue de littérature qui se focalise sur les travaux existants traitant le problème de disponibilité de ressources dans l'infonuage. Ce chapitre présente, aussi, une étude comparative entre notre solution et les travaux existants.

Le troisième chapitre présente les détails de la solution proposée. Nous commençons par introduire l'architecture du système de gestion des services infonuagiques proposé tout en

discutant ses composantes. Nous présentons, par la suite, une formulation mathématique du problème étudié ainsi que les algorithmes heuristiques proposés.

Dans le quatrième chapitre, nous décrivons les résultats expérimentaux. Nous commençons par la présentation de l'environnement expérimental et les scénarios considérés. Ensuite, nous évaluons l'efficacité des algorithmes proposés. Nous terminons ce rapport par une conclusion générale et nous identifions quelques perspectives pour les futurs travaux.

CHAPITRE 1

NOTIONS DE BASE

1.1 Introduction

Nous allons exposer dans ce premier chapitre les notions de base qui nous mènera à bien comprendre le problème étudié dans ce projet de recherche. Nous allons, ainsi, présenter les concepts reliés à notre domaine de recherche soit l'infonuagique, le réseau défini par logiciel, la virtualisation. Nous terminons par définir quelques notions relatives au problème de la disponibilité des ressources.

1.2 Infonuagique

Dans cette section, nous présentons le concept de l'infonuagique, ses différentes caractéristiques, les différents types de services et les modèles du déploiement de l'infonuage.

1.2.1 Définition

Le cloud computing ou l'informatique en nuage ou encore l'infonuagique est une technologie qui permet un accès sur demande, à travers un réseau Internet, à un pool de ressources informatiques configurables (réseaux, serveurs, stockage, applications et services) comme autant de « services » (Mell, 2011)(informatique, 2010)

1.2.2 Caractéristiques

Les principales caractéristiques de l'infonuage sont l'accès aux services à la demande, le paiement à l'usage, un large accès au réseau, la mutualisation et l'élasticité (Mell, 2011) :

- **Accès aux services à la demande :** L'utilisateur met en place et gère la configuration à distance au moyen d'une console de commande sans avoir nécessairement aucune interaction

humaine avec chaque fournisseur de services. Dans l'infonuage, la demande est automatique et la réponse est immédiate ;

- **Paiement à l'usage :** Les systèmes infonuagiques surveillent, contrôlent et rapportent l'utilisation des ressources, offrant une transparence au fournisseur aussi qu'au consommateur du service pour des raisons de facturation ;
- **Large accès au réseau :** Les services de l'infonuage sont mis à disposition sur Internet et utilisent des techniques standardisées permettant de servir les consommateurs qui se disposent d'un ordinateur, un téléphone ou une tablette ;
- **Mutualisation :** Les ressources hétérogènes (matériel, logiciel) sont combinées afin de servir plusieurs utilisateurs auxquels les ressources sont automatiquement attribuées ;
- **Élasticité :** Elle permet d'adapter les ressources aux variations de la demande d'une manière automatique. Les ressources peuvent être, donc, approvisionnées et libérées de manière élastique ;

1.2.3 Types de services infonuagiques

Le Cloud Computing peut être représenté en trois services principales dont l'utilisation diffère selon les besoins (informatique, 2010). Certains appellent ces services les couches de nuages et les représentent souvent sous forme d'une pyramide comme l'illustre la Figure 1.1 (Mell, 2011) :

- **IaaS (Infrastructure as a service) :** c'est la couche inférieure qui représente les ressources matérielles (puissance de calcul, espace de stockages , serveur, etc.). Les clients à ce niveau n'ont pas de contrôle sur ces ressources, mais plutôt sur les systèmes d'exploitation et les applications qu'ils peuvent installer sur le matériel alloué. En d'autres termes, une entreprise pourra par exemple louer des serveurs Linux, Windows ou autres systèmes, qui tourneront en fait dans une machine virtuelle chez le fournisseur de l'IaaS ;
- **PaaS (Plateform as a Service) :** c'est la couche intermédiaire où le fournisseur contrôle le système d'exploitation et les outils d'infrastructure et par suite, les clients n'ont le contrôle

que sur les applications déployées. En d'autres termes, le fournisseur offre au client un environnement fonctionnel et prêt à l'emploi comme l'exemple d'un environnement de développement et de test ;

- **SaaS (Software as a service) :** c'est la couche supérieure qui correspond à la mise à disposition des applications comme étant des services accessibles via Internet et non pas des composants techniques. Les clients n'ont plus besoin, donc, d'installer leurs applications sur ses postes.

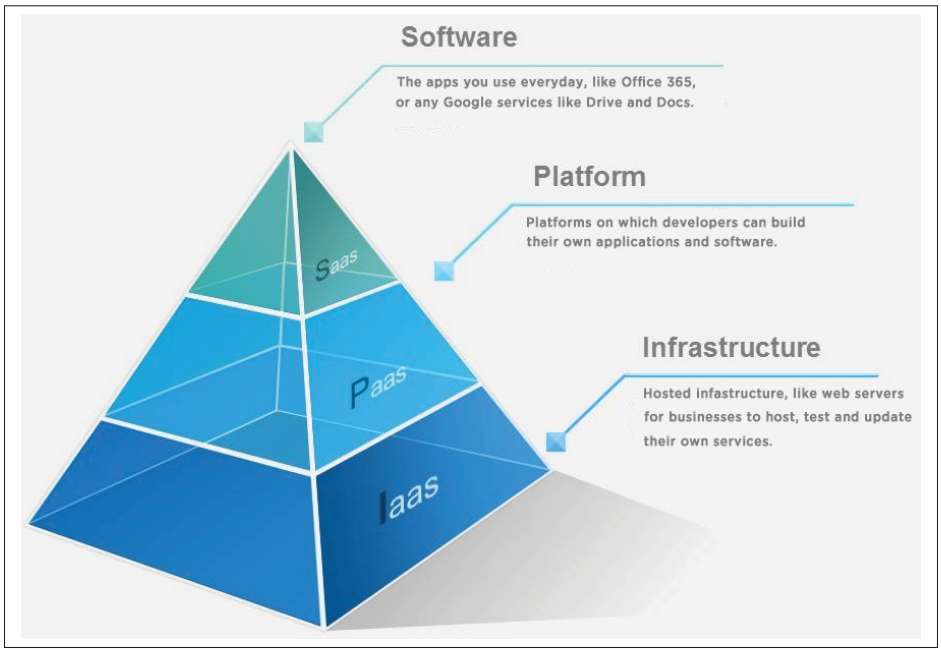


Figure 1.1 Les services de l'infonuagique (Technologies).

1.2.4 Modèles de déploiement du nuage

La littérature identifie quatre modèles de déploiement représentant les différentes manières dont l'infonuage peut être implémenté (Mell, 2011).

• **Nuage public :** C'est le nuage dans le sens traditionnel où les services sont offerts au grand public et peuvent être payants ou même gratuits. Ce cloud est, donc, externe à l'organisation,

accessible via Internet, géré par un prestataire externe propriétaire des infrastructures, avec des ressources partagées entre plusieurs sociétés.

• **Nuage privé :** Ce nuage est destiné entièrement pour une utilisation exclusive par un organisme. Il est accessible via des réseaux sécurisés. Le cloud privé peut être interne où il est hébergé et géré par l'organisme ou bien externe où il est hébergé et géré par un tiers et accessible via des réseaux de type VNF.

• **Nuage hybride :** C'est la combinaison du nuage public et nuage privé par une entreprise amenée à « coopérer », à partager entre eux applications et données.

• **Nuage communautaire :** Un ensemble d'organisation qui a un intérêt commun partage l'infrastructure du nuage. Il est plus couteux que le nuage public, mais offre d'autres avantages. Il peut être possédé, géré et exploité par une ou plusieurs organisations de la communauté, par un tiers, ou par une combinaison de ceux-ci, et il peut exister sur place ou hors site.

1.3 La réseautique définie par logiciel – SDN

Nous allons présenter, dans cette section, le concept de la réseautique définie par logiciel, son architecture et ses caractéristiques.

1.3.1 Définition

Selon l'Open Network Foundation (ONF), la réseautique définie par logiciel est une architecture émergente dynamique et gérable qui transforme la façon dont les infrastructures des réseaux informatiques sont gérées, contrôlées et configurées. Cette architecture dissocie les fonctions de contrôle de réseau et de transmission, ce qui permet à la commande de réseau de devenir directement programmable et de résumer l'infrastructure sous-jacente pour les applications et les services réseau.»

Le concept de la SDN repose sur la séparation des tâches de contrôle du réseau et la tâche de transmission entre deux plans séparés : plan de contrôle et plan de données. Le plan de contrôle

représente l'intelligence du réseau grâce à un contrôleur responsable de toutes les fonctions de gestion et de contrôle à savoir la prise des décisions de routages. Ainsi, les administrateurs sont capables de configurer et gérer automatiquement le réseau à l'aide des interfaces de programmation applicatives (Application Programming Interface - API).

Le plan de données est constitué d'un ensemble de commutateurs qui sont en charge du transfert des paquets à travers le réseau en se basant sur des règles de commutations. Ces règles sont préalablement installées dans les tables de commutation des commutateurs par le contrôleur à l'aide d'un protocole de communication standardisé à savoir OpenFlow, NetConf et ForCES.

1.3.2 Caractéristiques de la technologie SDN

Parmi les principales caractéristiques de la SDN, selon (CISCO), nous citons la programmabilité, l'homogénéité, l'agilité et la gestion centralisée :

- **La programmabilité :** Afin d'automatiser le réseau, les équipements réseaux peuvent être programmés par les administrateurs en utilisant des APIs ;
- **L'homogénéité :** La standardisation du protocole de communication permet de résoudre le problème d'interopérabilité. Par conséquent, les administrateurs ont la liberté d'utiliser de différentes marques des composants matériels ;
- **L'agilité :** La séparation du plan de contrôle et le plan de données permettent d'adapter dynamiquement les flux de trafic pour rencontrer les besoins du réseau en changement ;
- **La gestion centralisée :** L'intelligence du réseau est centralisée dans les contrôleurs SDN pour maintenir une vue globale du réseau.

1.4 Virtualisation

Nous présentons, dans cette section, le concept de la virtualisation et la virtualisation des fonctions de réseau.

1.4.1 Définition

La virtualisation est l'abstraction du matériel physique afin de faire fonctionner plusieurs machines virtuelles (MVs) au sein d'une même machine physique. Chaque machine virtuelle agit comme une machine physique ayant son propre système d'exploitation.

Ainsi, la virtualisation fait intervenir trois principaux composants :

- un système d'exploitation principal installé sur la machine physique, appelé système hôte, car il joue le rôle d'hôte à d'autres systèmes d'exploitation ;
- un hyperviseur qui est un outil de virtualisation installé sur le système hôte qui fournit l'environnement dans lequel différentes machines virtuelles s'exécutent ;
- un système d'exploitation installé dans une machine virtuelle, appelé système invité, qui fonctionne indépendamment des autres systèmes invités dans d'autres machines virtuelles.

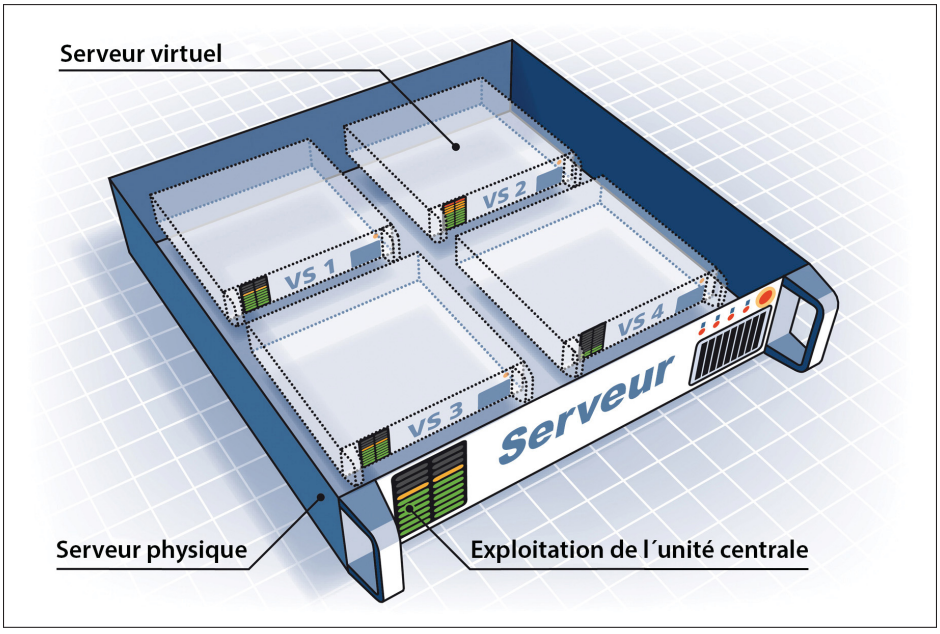


Figure 1.2 Virtualisation des serveurs (STRATO).

1.4.2 Virtualisation des fonctions réseau

La virtualisation des fonctions de réseau (NFV) est un nouveau paradigme architectural proposé afin d'améliorer la flexibilité de la fourniture de services réseau et réduire le délai de mise sur le marché de nouveaux services.

L'idée principale du NFV est le découplage des équipements de réseau physiques des fonctions qui les exécutent. Cela signifie qu'une fonction réseau, telle qu'un routeur, est instanciée comme étant une instance de logiciel standard appelée une fonction réseau virtuelle (Virtual Network Function - VNF). En d'autres termes, une fonction réseau virtuelle est une implémentation d'une fonction réseau (Network Function - NF) sous forme d'une machine virtuelle ou un conteneur. De cette façon, un service donné peut être décomposé en un ensemble de fonctions de réseau virtuel (VNFs), qui pourraient ensuite être implémentées dans des logiciels s'exécutant sur un ou plusieurs serveurs physiques standard. Par conséquent, il est maintenant possible de créer de manière dynamique des chaînes de services réseau (chaînes de fonctions de service - SFCs) capables de traiter le trafic entrant et de le diriger à travers une chaîne de fonctions de réseau virtuel (VNF) telles que des routeurs, des IDSs et des NATs exécutés sur des machines virtuelles.

Pour résumer, la virtualisation des fonctions de réseau a révolutionné la façon dont les fournisseurs de services conçoivent, déploient et gèrent leurs services. En effet, NFV promet aux fournisseurs de services une plus grande flexibilité de déployer leurs services réseau plus rapidement et à moindre coût, de manière à améliorer l'agilité de ces services grâce à un certain nombre de différences dans la manière dont la fourniture de services réseau est réalisée par rapport à la pratique actuelle (Mijumbi *et al.*, 2016)

- découpler le logiciel du matériel : la séparation des fonctions réseau des équipements qui les exécutent permet l'évolution des deux indépendamment de l'autre ;
- déploiement flexible de la fonction réseau : le détachement du logiciel du matériel permet de réaffecter et de partager les ressources d'infrastructure. Par conséquent, les composants peuvent être instanciés sur n'importe quel périphérique du réseau et leurs connexions

peuvent être configurées de manière flexible. Cela aide les opérateurs réseau à déployer plus rapidement de nouveaux services réseau sur la même plate-forme physique ;

- mise à l'échelle dynamique : Le découplage des fonctionnalités de la fonction réseau en composants logiciels instanciables offre une plus grande flexibilité pour adapter les performances des VNFs au besoin de l'opérateur de manière dynamique.

La figure 1.3 présente l'architecture de la virtualisation des fonctions réseau qui se compose selon (ETSI, 2015b) de trois composantes : l'infrastructure de la virtualisation de fonction de réseau (Network Function Virtualisation Infrastructure - NFVI), les fonctions de réseau virtuel (VNFs) et la gestion du NFV et l'orchestration du réseau (NFV management and network orchestration - NFV MANO).

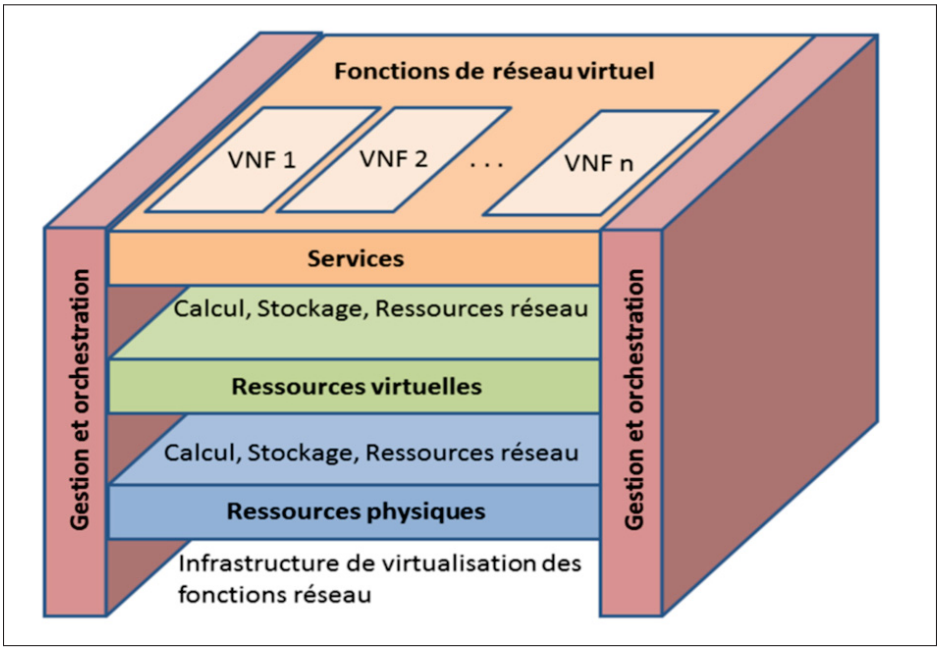


Figure 1.3 Architecture de la virtualisation des fonctions réseau (Mijumbi *et al.*, 2016).

- **L'infrastructure de la virtualisation de fonction de réseau (NFVI)** : NFVI est une combinaison des ressources matérielles (des matériels informatiques) et logicielles (des abstractions

des ressources informatiques, de stockage et réseau) qui constituent l'environnement dans lequel les VNFs sont déployés ;

- **Les fonctions de réseau virtuel (VNFs)** : C'est une implémentation d'une fonction réseau qui est déployée sur des ressources virtuelles telles qu'une machine virtuelle ou un conteneur ;
- **NFV MANO** : NFV MANO fournit les fonctionnalités de management et d'orchestration requises pour l'approvisionnement de VNF.

1.5 Conclusion

Ce chapitre présente les concepts de base liés à l'infonuagique, la réseautique définie par logiciel et la virtualisation des fonctions réseau. Nous avons décrit les principaux types et caractéristiques de chacun de ces concepts. Le chapitre suivant présente une revue de la littérature des approches utilisées pour la gestion de la disponibilité des ressources dans l'infonuage.

CHAPITRE 2

REVUE DE LA LITTÉRATURE SUR LA GESTION DE LA DISPONIBILITÉ DES RESSOURCES DANS L'INFONUAGE

2.1 Introduction

Ce chapitre est consacré à l'étude des différents travaux existants qui portent sur le problème de disponibilité des ressources dans le cloud en discutant leurs limites.

Les propositions existantes visant à gérer les défaillances et à les mitiger peuvent être globalement classées en techniques réactives et proactives Zhani & Boutaba (2015). Dans les techniques proactives, les VNFs de sauvegarde sont configurés chaque fois qu'un SFC est reçu et incorporé. Ces sauvegardes restent inactives, mais ne sont activées qu'en cas d'échec pour remplacer les VNFs défaillants afin de reproduire le service Ayoubi, Chen & Assi (2016b) ; Rabbani, Zhani & Boutaba (2014) ; Rahman & Boutaba (2013) ; Yu, Anand, Qiao & Sun (2011).

La seconde catégorie de solutions existantes est les techniques réactives. Ces techniques ne pré-allouent pas les ressources de sauvegarde, mais traitent les défaillances après leur apparition Bo, Huang, SUN, CHEN & LIU (2014) ; Rahman & Boutaba (2013) ; Xiao, Wang, Meng, Qiu & Li (2014). Par conséquent, elles ont besoin de plus de temps pour allouer des ressources et mettre en service de nouvelles instances de VNFs afin de prendre en charge le service. Cela entraîne certainement une interruption de service plus longue, ce qui est très coûteux pour les fournisseurs de services Zhani & Boutaba (2015). Cela rend les techniques proactives plus attrayantes même si elles gaspillent des ressources pour les VNFs de sauvegarde.

Pour remédier à ce problème et minimiser le gaspillage de ressources, plusieurs efforts de recherche préconisent l'utilisation des VNFs de sauvegarde partagés, Ayoubi *et al.* (2016b) dans lesquels la même ressource de sauvegarde peut être utilisée pour mitiger la défaillance d'un ensemble de VNFs, en supposant qu'ils n'échouent pas simultanément (c'est-à-dire qu'un seul VNF de cet ensemble peut échouer à la fois). Dans ce travail, nous proposons des solutions pour

assurer la disponibilité des chaînes de service contre les défaillances d'un seul nœud physique en utilisant des ressources de sauvegarde partagées. Contrairement aux travaux antérieurs traitant le même problème où les sauvegardes sont partagées uniquement entre les VNFs de la même chaîne Yu *et al.* (2011) Ayoubi *et al.* (2016b), notre solution suppose que les VNFs de sauvegarde sont partagés entre toutes les chaînes intégrées dans l'infrastructure. Cela réduit considérablement la quantité de ressources utilisées pour les VNFs de sauvegarde tout en garantissant que tous les SFCs sont protégés contre les défaillances.

Dans ce qui suit, nous fournissons un aperçu des travaux abordant le problème de gestion de la disponibilité des ressources dans l'infonuage.

Nous terminons cette section avec une étude comparative qui met en comparaison nos solutions proposées à ceux qui existent déjà en fonction de leurs objectifs ciblés.

2.2 Méthodes de gestion de disponibilité des ressources dans l'infonuage

2.2.1 Méthodes réactives

Bo *et al.* (2014) ont proposé un algorithme qui, en cas de défaillance de la liaison, cherche d'autres ressources pour réattribuer le chemin de bout en bout ou réintégrer l'ensemble du réseau virtuel si les ressources ne sont pas suffisantes. Cela peut entraîner un temps de convergence long et des temps d'arrêt de service plus longs.

Ayoubi, Assi, Narayanan & Shaban (2016a) ont démontré la nature NP-difficile de l'incorporation des SFCs en tenant compte de la capacité de survie et ont proposé un algorithme heuristique pour restaurer les services défaillants tout en maintenant les exigences de qualité de service en termes de délais en cas de défaillance d'un nœud unique. Plusieurs défaillances ont été traitées dans Ghaleb, Khalifa, Ayoubi, Shaban & Assi (2016) où un algorithme a été introduit afin de trouver un nœud de sauvegarde. L'algorithme repose sur des techniques de filtrage permettant d'analyser l'espace de la solution et d'accélérer le processus de recherche des sauvegardes.

Karra & Sivalingam (2018) ont abordé le problème de la défaillance d'un ou de plusieurs nœuds physiques. Les auteurs ont proposé une architecture de résilience dans laquelle les nœuds envoient des messages de maintien en activité (Keep-alive) au contrôleur pour indiquer qu'ils sont opérationnels. En cas de défaillance d'un ou plusieurs nœuds, le contrôleur lance un algorithme en deux étapes pour restaurer le SFC. La première étape consiste à trouver une solution réalisable avec un MTTF minimum en utilisant l'algorithme de Yen's K-shortest Path en utilisant le prédécesseur et le successeur des fonctions défaillantes en tant que source et destination. Si un chemin est trouvé, le contrôleur migre les fonctions vers les nouveaux serveurs sur ce chemin. Dans la deuxième étape, cette solution est, ensuite, transmise à l'algorithme Tabu Search, qui tente de trouver une meilleure solution avec moins de MTTF que celle fournie par l'algorithme Yen's K-shortest Path. Si un nouveau chemin est trouvé, les fonctions sont migrées vers les serveurs de ce chemin. La principale limite ici est le nombre élevé de migrations coûteuses pour les fournisseurs de services.

2.2.2 Méthodes proactives

Yu *et al.* (2011) ont examiné le cas d'une défaillance d'un nœud et ont présenté deux approches en deux étapes pour l'approvisionnement de nœuds de sauvegarde. La première étape consiste à concevoir une demande VN améliorée en ajoutant des nœuds virtuels et des liens de sauvegarde, tandis que la deuxième étape place l'EVN dans le réseau. La première approche, appelée 1-redondant, reconfigure la requête de réseau virtuel en une requête pouvant survivre en ajoutant un nœud de sauvegarde unique et des connexions redondantes. La seconde approche s'appelle k-redundant, où k est une constante qui représente le nombre de nœuds de sauvegarde à approvisionner, dans lequel chaque nœud critique est autorisé à avoir un nœud de sauvegarde correspondant. Pour minimiser la bande passante lors du placement de la requête résultante, les auteurs ont introduit des techniques de partage de sauvegarde et de partage croisé. Le partage de sauvegarde désigne le partage de la bande passante entre différents chemins de sauvegarde et le partage croisé signifie la réutilisation d'un lien libéré en raison de la défaillance du nœud. Le problème de ces approches est qu'un seul nœud redondant risque de ne pas suffire, alors

que k nœuds redondants risquent d'entraîner un gaspillage de ressources. Pour remédier à cette limitation, les solutions présentées dans ce travail visent à trouver le nombre optimal de sauvegardes lorsque le ILP proposé est utilisé ou du moins à le minimiser lorsque les heuristiques proposées sont utilisées.

Dans le même sens, Ayoubi *et al.* (2016b) a examiné le cas d'une défaillance d'un seul nœud et a proposé à ProRed, une nouvelle technique qui améliore les réseaux virtuels en ajoutant des noeuds de sauvegardes. ProRed va au-delà des techniques qui fixent le nombre de nœuds de sauvegarde à 1 ou k et explore, plutôt, l'espace entre 1 et k pour trouver le nombre optimal de nœuds de sauvegarde à incorporer au réseau virtuel demandé. À l'instar de Yu *et al.* (2011), ayoubi et al ont tenté d'exploiter à la fois le partage de sauvegarde et le partage croisé. Toutefois, dans cette solution, les nœuds virtuels de sauvegarde sont configurés pour chaque demande et ne sont donc pas partagés avec d'autres réseaux virtuels. Notre travail est différent puisque les sauvegardes sont partagées entre tous les nœuds virtuels appartenant à toutes les SFCs intégrées dans l'infrastructure physique. En conséquence, nos solutions réduisent davantage le nombre total de sauvegardes dans le système.

S'attaquant aux défaillances de plusieurs nœuds physiques, Xiao *et al.* (2014) ont proposé une solution tenant compte de la topologie pour assurer la survie des réseaux virtuels. Cette solution comprend trois phases. Dans la première phase, avant l'arrivée d'une requête de réseau virtuel, un ensemble de candidats est calculé pour chacun des nœuds physiques et dédié à des fins de sauvegarde. Ces candidats peuvent être définis comme les nœuds pouvant atteindre tout ce qui entoure le nœud physique dans une certaine zone géographique. À la deuxième phase, lors de la réception d'une requête, une incorporation de VN basée sur la capacité de récupération est utilisée pour allouer le VN, de telle sorte que les noeuds virtuels critiques sont placés dans des nœuds physiques qui ont les meilleurs candidats. La phase finale est déclenchée en cas d'échec. Un algorithme de réallocation est utilisé pour récupérer autant de VN que possible. Si les ressources de sauvegarde sont limitées, les VN ayant une pénalité plus élevée sont priorisés. Cet algorithme de réallocation a été formulé en tant que ILP dans le but de minimiser la pénalité totale encourue.

Rahman & Boutaba (2013) ont également proposé une approche hybride qui tire parti d'un ensemble de détours de secours possibles pour chaque lien. Ces détours sont précalculés de manière proactive avant l'arrivée des demandes de réseaux virtuels pour permettre un réacheminement rapide en cas de défaillance de la liaison. En d'autres termes, avant qu'une requête VN n'arrive, un algorithme de sélection de chemin calcule les chemins de sauvegarde pour chaque lien de substrat. En cas de défaillance, un mécanisme d'optimisation en ligne et réactif redirige la bande passante de la liaison défaillante vers le chemin de sauvegarde. Ce mécanisme de sauvegarde est une approche de restauration. Donc, après une panne, il ne peut pas garantir une récupération à 100%. Un autre problème est que, dans un réseau physique très saturé, les ressources nécessaires pour les sauvegardes peuvent ne pas être disponibles, car la bande passante peut être utilisée pour les nouvelles demandes et il peut ne pas rester suffisamment de ressources pour la récupération. Par conséquent, dans un réseau physique très saturé, le mécanisme de sauvegarde peut ne pas restaurer le VN et une grande quantité de données peut être perdue.

Xu, Tang, Kwiat, Zhang & Xue (2012) ont proposé un système d'optimisation pour l'approvisionnement des centres de données virtualisés (VDC) avec des VM et des liens de sauvegarde. Plusieurs machines virtuelles et leurs sauvegardes sont regroupées pour former une infrastructure virtuelle survivant (SVI) pour un service. Leur objectif est de placer l'interface SVI sur le réseau du substrat tout en minimisant la bande passante de secours inactive et en garantissant une restauration sans interruption ni dégradation. Les auteurs ont divisé le problème en deux sous-problèmes : les problèmes de VM Placement (VMP) et de Virtual Link Mapping (VLM) pouvant être résolus séparément. Pour le sous-problème de placement de machine virtuelle, basé sur la recherche Profondeur en premier, les auteurs ont conçu une heuristique de retour en arrière qui produit plusieurs solutions VMP qui représentent un ensemble de solutions de mappage de nœuds réalisables. Ces solutions sont transmises à un algorithme basé sur un programme linéaire (LP) (LP-VLM) pour résoudre le sous-problème de mappage de lien virtuel pour chaque solution VMP et choisir la solution avec la bande passante de sauvegarde réservée la plus basse. En outre, un mappage coordonné de nœud virtuel et de lien virtuel a été introduit. Cet algorithme

de mappage conjoint détermine une paire de serveurs pour chaque lien virtuel et alloue la bande passante entre un LP et une résolution ultérieure du LP-VLM. Pour le problème VMP, un assez grand nombre de solutions possibles sont calculées, même lorsqu'il est restreint, et encore une fois, pour toutes les solutions VMP possibles, la VLM doit être calculée. Les principales limites ici sont d'abord la surcharge informatique importante des grands réseaux et, deuxièmement, les auteurs n'ont pas pris en compte la disponibilité des machines physiques et des liaisons.

2.3 Comparaison entre les méthodes existantes

Nous résumons dans le tableau 2.1 les solutions couvertes dans la revue de la littérature. Le tableau présente le type de la solution (réactif ou proactif) et indique si elle adresse une ou plusieurs pannes, les pannes de nœuds ou de liaisons et si la sauvegarde est partagée entre les nœuds virtuels de tous les réseaux virtuels ou entre les nœuds virtuels d'un seul réseau virtuel.

Comme le montre le tableau 2.1, la nouveauté de notre travail réside dans l'idée de partager les sauvegardes entre des VNFs de même type appartenant à des chaînes de services différentes plutôt qu'à la même chaîne de services. Ceci permet de réduire la quantité de ressources de sauvegarde tout en garantissant la pérennité des réseaux virtuels à chaque échec.

Tableau 2.1 Solutions existantes par rapport aux solutions proposées.

Solutions	Type de la solution		Nombre de défaillance		Type de la défaillance		Prise en charge des sauvegardes partagées		
	Proactive	Reactive	Unique	Multiple	Noeud	Lien	Supporté	Partagé entre tous les VN	Partagée entre un seul VN
Yu *et al.* (2011)	X		X		X		X		X
Ayoubi *et al.* (2016b)	X		X		X		X		X
Xiao *et al.* (2014)	X			X	X			-	-
Rahman & Boutaba (2013)	X		X			X		-	-
Xu *et al.* (2012)	X		X		X			-	-
Bo *et al.* (2014)		X	X			X		-	-
Ayoubi *et al.* (2016a)		X	X		X			-	-
Ghaleb *et al.* (2016)		X		X	X	X		-	-
Karra & Sivalingam (2018)		X		X	X			-	-
Solutions proposées	X		X		X		X	X	

2.4 Conclusion

Ce chapitre présente une étude approfondie des travaux qui ont abordé le problème de la gestion de la disponibilité des ressources dans l'infonuage. Ces solutions sont classées en solutions réactives et proactives. Généralement, les solutions réactives entrainent une longue interruption de service par rapport aux solutions proactives dont les limites sont le gaspillage de ressources sous forme des sauvegardes. Contrairement à ces solutions, nos solutions proposées visent à minimiser la quantité de ressources gaspillées en considérant le partage des sauvegardes entre des VNFs de même types.

Le chapitre suivant détaille nos solutions proposées. Nous présentons, en premier lieu, notre formulation mathématique du problème en question et en deuxième lieu, les algorithmes développés pour atteindre nos objectifs.

CHAPITRE 3

FORMULATION MATHÉMATIQUE ET SOLUTIONS PROPOSÉES

3.1 Introduction

Dans ce chapitre, nous présentons, d'abord, l'architecture du système de gestion proposé qui intègre la gestion de la disponibilité des chaînes de service et nous discutons ses composantes principales proposées afin d'atteindre nos objectifs à savoir assurer la continuité des services affectés par une défaillance.

Nous présentons, ensuite, une modélisation mathématique de ce problème à l'aide d'un Programme Linéaire en Nombres Entiers (PLNE) qui permet de minimiser le nombre des VNFs de sauvegarde en les plaçant dans des emplacements optimaux. Enfin, nous introduisons cinq algorithmes heuristiques pour la résolution du problème à grande échelle.

3.2 Architecture du mécanisme de gestion de la disponibilité

Afin d'assurer la disponibilité des ressources des chaînes de service mappées dans une infrastructure NFVI, nous avons proposé un module de disponibilité incorporée dans un mécanisme de gestion. La figure 3.1 montre les principales composantes de ce mécanisme.

• **Module d'approvisionnement des chaînes de service (Service Chain Provisioning Module) :** Ce module alloue les ressources pour les chaînes de service et instancie les machines virtuelles requises exécutant les fonctions réseau. Il utilise également le contrôleur SDN pour fournir la quantité de bande passante requise et pour définir les règles de transfert requises dans les commutateurs afin de diriger le trafic sur les VNFs composant chaque chaîne de services. Il est à noter que la conception de ce module est en dehors de la portée de ce travail. Il existe de nombreux travaux sur ce module et toutes les solutions existantes pourraient être utilisées. (Par exemple, Herrera & Botero (2016) ; Luizelli *et al.* (2015) ; Racheg *et al.* (2017)).

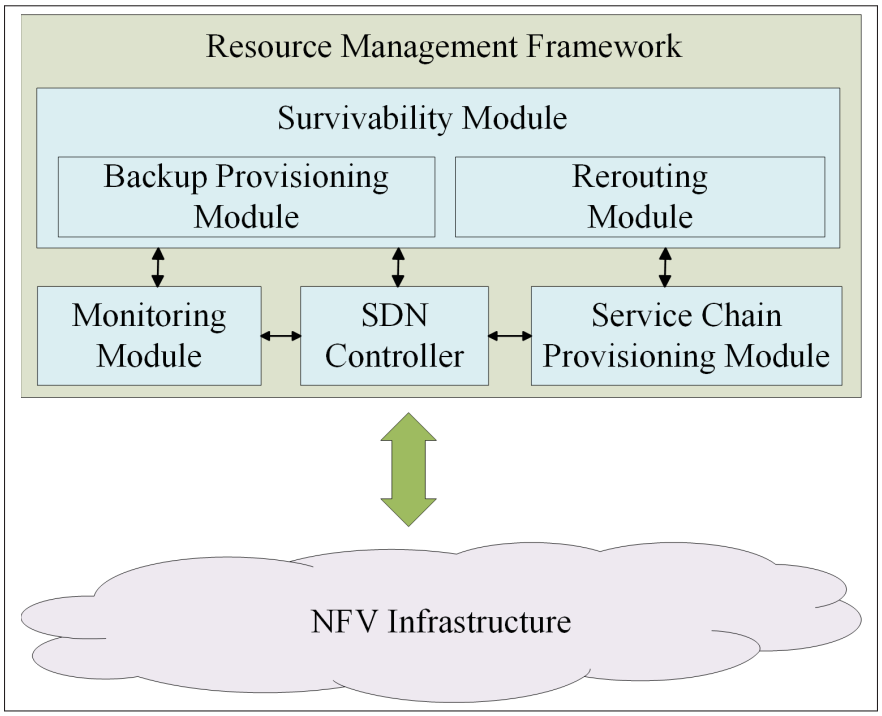


Figure 3.1 Architecture du système de gestion des ressources proposé avec le module de disponibilité.

• **Module de surveillance (Monitoring Module) :** Ce module est chargé de surveiller en permanence les nœuds et les liens physiques de l'infrastructure et d'envoyer en temps réel l'état des ressources aux autres modules. Quand une panne est détectée, le module de surveillance le rapporte au module de disponibilité qui, à son tour, réagit pour surmonter la défaillance et assurer la continuité du service.

• **Module de réacheminement (Rerouting Module) :** En cas d'échec, ce module redirige le trafic destiné à l'origine aux VNFs défaillants, vers les VNFs de sauvegarde.

• **Module de disponibilité (Backup Provisioning Module) :** Ce module est chargé d'identifier le nombre minimal de sauvegardes nécessaires pour garantir la disponibilité des chaînes intégrées et de déterminer leurs emplacements. Ce module instancie également les VNFs de sauvegarde et les liens de synchronisation nécessaires pour les maintenir à jour.

Dans la section suivante, nous nous concentrons sur la conception du module de disponibilité et nous décrivons en détail les solutions proposées pour approvisionner des VNFs de sauvegarde.

3.3 Formulation mathématique du problème

Dans cette section, nous formulons le problème de la disponibilité des chaînes de services comme un Programme Linéaire en Nombres Entiers (ILP) visant à déterminer le nombre et l'emplacement des VNFs de sauvegarde afin de réduire au minimum la quantité de ressources allouées aux instances de sauvegarde tout en garantissant un coût et un délai de synchronisation minimaux.

• Modélisation de l'infrastructure et des chaînes de service

Nous modélisons l'infrastructure physique, composée d'un ensemble de nœuds géographiquement distribués, sous la forme d'un graphe noté $G = (N, L)$, où N est l'ensemble des nœuds physiques et L l'ensemble des liens physiques qui les relient.

Chaque nœud physique $n \in N$ a une capacité c_n, exprimée par le nombre maximal de VNFs pouvant être hébergés par le nœud physique $n \in N$. Pour simplifier, nous supposons que chaque VNF s'exécute sur une seule machine virtuelle de taille standard. Nous pouvons, donc, voir c_n comme le nombre maximal de machines virtuelles pouvant être approvisionnées dans le nœud physique n. Nous définissons également d_{in} comme le nombre minimal de sauts séparant les nœuds physiques i et n.

Nous modélisons la chaîne de service sous forme de graphique noté $S = (V, E)$ où V est l'ensemble des VNFs qui le composent et E l'ensemble des liens virtuels qui les connectent. Nous supposons qu'il existe différents types de VNFs (par exemple pare-feu, IDS, NAT). On note J l'ensemble des types de VNF et on définit m_{ij} comme le nombre de VNFs de type $j \in J$ incorporés dans le nœud physique i.

Tableau 3.1 Tableau de notations.

Réseaux	
N	l'ensemble des noeuds physiques
L	l'ensemble des liens physiques
c_n	la capacité du noeud n
d_{in}	la distance entre deux noeuds physiques n et i
Chaînes de services	
V	l'ensemble des VNFs constituant la chaînes
E	l'ensemble des liens virtuels de la chaîne
J	l'ensemble des types de VNF
m_{ij}	le nombre de VNFs de type j placé dans le noeud physique i
Variables	
x_{ij}	le nombre des VNFs de sauvegarde de type j intégrés au noeud physique i
y_{ijn}	1, si les sauvegardes approvisionnées pour les VNFs de type j intégrés dans le noeud physique i sont hébergées dans le noeud physique n et 0, sinon.

•Variables de décision

Nous définissons deux variables de décision. On note x_{ij} le nombre des VNFs de sauvegarde de type j placés au nœud physique i. Nous définissons également $y_{ijn} \in \{0,1\}$ pour indiquer si les sauvegardes approvisionnées pour les VNFs de type j placés dans le nœud physique i sont hébergées dans le nœud physique n.

•Contraintes

Afin de trouver une solution réalisable, plusieurs contraintes doivent être satisfaites. Par exemple, pour s'assurer que les VNFs principaux et de sauvegarde ne sont pas incorporés dans le même nœud physique, la contrainte suivante doit être satisfaite :

$$y_{iji} = 0 \quad \forall i \in N, \forall j \in J \tag{3.1}$$

Nous devons également nous assurer que tous les VNFs de type j placés dans le nœud physique i disposent nécessairement de sauvegardes dans un ou plusieurs nœuds physiques :

$$\sum_{n \in N} y_{ijn} \geq 1 \quad \forall i \in N, \forall j \in J \tag{3.2}$$

De plus, si les sauvegardes des VNFs de type j placés dans le nœud physique i sont hébergés dans le nœud physique n, le nombre total de sauvegardes de type j configurées dans n doit être supérieur ou égal au nombre des VNFs de type j placés dans le nœud i. En d'autres termes, nous avons :

$$\forall \quad y_{ijn} = 1, \quad \sum_{n \in N} x_{nj} \geq m_{ij} \quad \forall i, n \in N, \forall j \in J \tag{3.3}$$

Cette instruction peut être traduite comme contrainte suivante :

$$m_{ij} \leq \sum_{n \in N} x_{nj} + M(1 - y_{ijn}) \quad \forall i, n \in N, \forall j \in J \tag{3.4}$$

Où M est une constante de grande valeur (autour de 10 000).

En outre, pour garantir que le nœud physique hébergeant les sauvegardes dispose de ressources suffisantes, la contrainte de capacité suivante doit être satisfaite pour chaque nœud physique n :

$$\sum_{j \in J} m_{nj} + \sum_{j \in J} x_{nj} \leq c_n \quad \forall n \in N \tag{3.5}$$

Où le premier terme représente le nombre de VNFs hébergés dans le nœud physique n et le second terme représente le nombre de sauvegardes hébergées dans le même nœud physique.

Enfin, comme nous devons minimiser les coûts de synchronisation et les délais entre les VNFs primaires et leurs sauvegardes, nous limitons le nombre de sauts entre chaque VNF et sa sauvegarde à un nombre limité de sauts dénotés par d_{max}. Ainsi, nous avons :

$$\forall \quad y_{ijn} = 1, \quad d_{in} \leq d_{max} \quad \forall i, n \in N, \forall j \in J \tag{3.6}$$

La déclaration précédente peut également être écrite comme suit contrainte :

$$d_{in} \leq d_{max} + M(1 - y_{ijn}) \quad \forall i, n \in N, \forall j \in J \tag{3.7}$$

Où M est une constante de grande valeur (autour de 10 000).

Pour simplifier, nous supposons que le nombre de sauts entre deux nœuds physiques reflète le délai entre eux. Cependant, cela peut ne pas être toujours vrai. Dans ce cas, notre modèle peut être facilement mis à jour pour prendre en compte le délai de propagation entre les nœuds en définissant d_{in} comme délai du chemin le plus court entre les nœuds i et n, et d_{max} comme délai maximal requis entre une VNF et sa sauvegarde.

•Fonction objectif

Notre objectif ultime est de minimiser la quantité de ressources utilisées par les VNFs de sauvegarde tout en satisfaisant toutes les contraintes susmentionnées. Cela peut être réalisé en minimisant le nombre total de sauvegardes dans tous les nœuds physiques de l'infrastructure physique. La fonction objective peut alors être écrite comme suit :

$$\min \sum_{i \in N} \sum_{j \in J} x_{ij} \tag{3.8}$$

3.4 Solutions proposées

Dans cette section, nous présentons les solutions conçues pour résoudre le problème de disponibilité des ressources. Ces solutions peuvent être classées en deux catégories : celles basées sur la migration et celles qui ne sont pas.

Pour la première catégorie, nous présentons trois algorithmes qui ne sont pas basés sur la migration.

- **Partage de sauvegarde «Pull» (Backup Sharing-Pull - BS-Pull)** où nous examinons chaque nœud physique pour trouver le nombre maximal de VNFs qu'il peut sauvegarder (nous appelons cela le tirage) ;
- **Partage de sauvegarde «Push» (Backup Sharing-Push - BS-Push)** où l'objectif est d'étendre la couverture d'un nœud physique afin de lui permettre d'héberger des sauvegardes de VNFs aussi dispersés que possible dans plusieurs nœuds physiques ;
- **Partage de sauvegarde « Distribué » (Distributed Backup Sharing - DBS)** qui tente de répartir les sauvegardes entre plusieurs hôtes.

Pour la deuxième catégorie, l'objectif de la migration est d'optimiser le rapport de partage. Ce ratio de partage est une métrique représentant le nombre total de VNFs principaux divisé par le nombre total de sauvegardes. Plus ce rapport est élevé, meilleur est l'algorithme d'approvisionnement, car cela signifie que nous utilisons un petit nombre de sauvegardes pour un grand nombre de VNFs. Pour ce faire, nous proposons deux algorithmes :

- **Migration Après BS-Pull (Migration After BS-Pull - MABS-Pull)** où nous migrons les VNFs qui n'ont plus de sauvegarde après l'application de l'algorithme BS-Pull ;
- **Partage de sauvegarde optimisé (Optimized Backup Sharing - OBS)** où nous migrons les VNFs avant et après l'application de l'algorithme afin d'optimiser le nombre de sauvegardes à allouer

Tableau 3.2 Comapraison des solutions proposées.

Solution	Migration	Objectif
BS-Pull	-	Trouver les noeuds que leurs sauvegardes peuvent être héberger dans un noeud n
BS-Push	-	Trouver le noeud n qui peut héberger des sauvegardes pour d'autres noeuds
DBS	-	Optimiser l'utilisation des ressources par distribuer les sauvegardes sur des noeuds différents
MABS-Pull	X	Minimiser le nombre de VNFs laissés sans sauvegardes
OBS	X	Optimiser le nombre de sauvegardes

3.4.1 Partage de sauvegarde Pull (Backup Sharing-Pull - BS-Pull)

BS-Pull vise à allouer les VNFs de sauvegarde pour chaque type de VNF. En supposant que nous considérions, d'abord, les VNFs de type $j \in J$, tous les nœuds de l'infrastructure physique sont supposés capables d'héberger des sauvegardes pour les VNFs de type j. Notre objectif dans les étapes suivantes est de sélectionner le nœud pouvant héberger ces sauvegardes et le nombre de sauvegardes par nœud.

Nous définissons, d'abord, l'ensemble des voisins source d'un nœud physique n (c'est-à-dire $SNeigh(n)$) comme l'ensemble des nœuds physiques pouvant être atteints à partir de n dans un nombre de sauts d_{max} et tel que le nœud n dispose de suffisamment de ressources pour héberger les sauvegardes requises pour sauvegarder des VNFs de type j hébergés dans l'un de ces voisins source. En d'autres termes, si les VNFs de sauvegarde sont configurés dans le nœud n, ils peuvent être partagés entre tous les voisins source de n.

Pour chaque nœud physique $n \in N$, nous calculons l'ensemble des voisins sources $SNeigh(n)$. Nous calculons également b_n, qui est le nombre de VNFs de type j profitant des sauvegardes pouvant être approvisionnées dans le nœud physique n (Lignes 9-19). Plus b_n est élevé, plus le nombre de VNFs partageant les sauvegardes est élevé. En conséquence, afin de maximiser le partage de sauvegarde, nous sélectionnons comme nœud hôte des sauvegardes, le nœud $n_{host} \in N$ qui a la plus grande valeur de b_n. Nous allouons ensuite les VNFs de sauvegarde dans le nœud n_{host} (fonction Allocate à la ligne 25). Nous répétons cette opération jusqu'à ce

Algorithm 1 BS-Pull

Inputs
N: set of physical nodes
u_n: total number of VNFs hosted in physical node n
m_{ij}: number of type j VNFs hosted in physical node i
for $j \in J$ **do** ▷ *Parsing VNF types*
  $BNodes(j) \leftarrow \emptyset$ ▷ *set of physical nodes whose type j VNFs have already backups*
  repeat
    ▷ *Parsing all potential hosting nodes*
    for $n \in N$ **do**
      $SNeigh(n) \leftarrow \emptyset$ ▷ *set of source neighbors for physical node n*
      $b_n \leftarrow 0$ ▷ *number of type j VNFs able to use shared VNF backups hosted in node n*
      ▷ *Finding source neighbors of n*
      for $i \in N \backslash (BNodes(j) \cup \{n\})$ **do**
        if $d_{ni} \leq d_{max}$ & $m_{ij} + u_n \leq c_n$ **then**
          $SNeigh(n) \leftarrow SNeigh(n) \cup \{i\}$
          $b_n \leftarrow b_n + m_{ij}$
        end if
      end for
    end for
    ▷ *Finding the node n_{host} that maximizes the number of type j VNFs that are backed up*
    $n_{host} = \arg\max_n b_n$
    ▷ *Compute the number of shared backups s*
    $s = \max_{i \in SNeigh(n_{host})}(m_{ij})$
    ▷ *Allocate backups and update $u_{n_{host}}$*
    Allocate (s backups, VNF type j, host n_{host})
    $BNodes(j) \leftarrow BNodes(j) \cup SNeigh(n_{host})$
  until $SNeigh(n) = \emptyset \quad \forall n \in N$
end for

Figure 3.2 Algorithme BS-Pull.

qu'aucun voisin source ne puisse être identifié pour tous les nœuds physiques. S'il n'a plus de voisin source pour tous les nœuds physiques, cela signifie qu'il n'y a pas assez de ressources pour héberger les VNFs de sauvegarde ou qu'il n'y a pas de VNFs de type j laissés sans sauvegardes. Ce processus est répété pour tous les types de VNF.

3.4.2 Partage de sauvegarde Push (Backup Sharing-Push - BS-Push)

Dans cet algorithme, nous adoptons une approche différente de celle du premier. Pour un nœud physique particulier, notre objectif est de maximiser le nombre de ses voisins sources qui l'utilisent (le nœud physique) pour héberger leurs sauvegardes (contrairement à BS-Pull qui optimise le nombre de VNFs sauvegardés par le nœud de sauvegarde, mais pas le nombre de voisins sources l'utilisant).

Algorithm 2 BS-Push

1: **Inputs**
2: N: set of physical nodes
3: u_n: total number of VNFs hosted in physical node n
4: m_{ij}: number of type j VNFs hosted in physical node i
5: **for** $j \in J$ **do** ▷ *Parsing VNF types*
6: $BNodes(j) \leftarrow \emptyset$ ▷ *set of physical nodes whose type j VNFs have already backups*
7: **repeat**
8: Compute $SNeigh(n) \quad \forall n \in N$
9: ▷ *Finding the node n_{host} that is connected to to the maximum number of neighbors*
10: $n_{host} = \arg\max_n |SNeigh(n)|$
11: ▷ *Compute the number of shared backups s*
12: $s = \max_{i \in SNeigh(n_{host})}(m_{ij})$
13: ▷ *Allocate backups and update $u_{n_{host}}$*
14: Allocate (s backups, VNF type j, host n_{host})
15: $BNodes(j) \leftarrow BNodes(j) \cup SNeigh(n_{host})$
16: **until** $SNeigh(n) = \emptyset \quad \forall n \in N$
17: **end for**

Figure 3.3 Algorithme BS-Push.

Similaire à BS-Pull, BS-Push calcule l'ensemble des voisins source pour tous les nœuds physiques (ligne 8). Cependant, l'algorithme sélectionne le nœud n_{host} qui a le plus grand nombre de voisins source afin d'héberger les sauvegardes de tous ces voisins (ligne 10). Les ressources de sauvegarde sont ensuite allouées et associées à tous les voisins du nœud sélectionné (ligne 14). L'opération est ensuite répétée jusqu'à ce qu'il n'y ait plus de voisins source pour tous les nœuds physiques. Enfin, l'ensemble du processus est appliqué à nouveau pour chacun des types de VNF.

3.4.3 Partage de sauvegarde distribué (Distributed Backup Sharing - DBS)

L'idée derrière cet algorithme est différente de celle des deux précédentes, bien qu'il repose également sur la recherche de voisins pour héberger des sauvegardes pour les VNFs principaux. DBS calcule l'ensemble des voisins source pour tous les nœuds physiques. Identique à BS-Push, l'algorithme sélectionne le nœud n_{host} qui contient le plus grand nombre de voisins source afin d'héberger les sauvegardes de tous ces voisins (ligne 10). Si le nœud n_{host} a une capacité suffisante pour héberger des sauvegardes pour tous ses voisins, toutes les sauvegardes seront alors allouées dans n_{host}, sinon nous n'allouons qu'un nombre de sauvegardes pouvant être hébergées dans n_{host}. Les ressources de sauvegarde sont, ensuite, allouées et l'opération est répétée jusqu'à ce qu'il n'y ait plus de voisins source pour tous les nœuds physiques et à chaque fois les voisins dont tous les VNFs sont sauvegardés ne sont pas pris en compte dans l'itération. Enfin, le processus est appliqué à nouveau pour chaque type de VNF.

Algorithm 3 DBS

1: **Inputs**
2: N: set of physical nodes
3: u_n: total number of VNFs hosted in physical node n
4: m_{ij}: number of type j VNFs hosted in physical node i
5: **for** $j \in J$ **do** ▷ *Parsing VNF types*
6: $BNodes(j) \leftarrow$ ▷ *set of physical nodes whose type j VNFs have already backups*
7: **repeat**
8: Compute $SNeigh(n) \quad \forall n \in N$
9: ▷ *Finding the node n_{host} that is connected to to the maximum number of neighbors*
10: $n_{host} = \arg\max_n |SNeigh(n)|$
11: ▷ *Compute the number of shared backups s*
12: $s = \min_{i \in SNeigh(n_{host})}(m_{ij}, c_n)$
13: ▷ *Allocate backups and update $u_{n_{host}}$*
14: Allocate (s backups, VNF type j, host n_{host})
15: $BNodes(j) \leftarrow BNodes(j) \cup SNeigh(n_{host})$
16: **until** $SNeigh(n) = \quad \forall n \in N$
17: **end for**

Figure 3.4 Algorithme DBS.

3.4.4 Migration Aprés BS-Pull (Migration After BS-Pull - MABS-Pull)

Algorithm 4 MABS-Pull

Inputs: Output of BS-Pull for each VNF type
N: set of physical nodes
u_n: total number of VNFs hosted in physical node n
m_{ij}: number of type j VNFs hosted in physical node i
$BNodes(j)$: set of physical nodes whose type j VNFs have already backups
$BHosts(j)$: set of physical nodes hosting type j backups
$x_{n_{host}}$: number of type j backups
c_n: remaining capacity in each node $n \in N$
for $j \in J$ **do** ▷ *Parsing VNF types*
  for $n \in N$ **do**
    $MigrateTo(n) \leftarrow$ ▷ *set of candidate nodes to host migrated VNFs from a physical node n*
    $b_n \leftarrow 0$ ▷ *number of type j VNFs to be migrated*

    ▷ *Finding candidate nodes to host migrated VNFs from n*
    for $i \in BNodes(j) \backslash (BHosts(j) \cup \{n\})$ **do**
      if $d_{ni} \leq d_{max}$ & $c_n \geq 0$ **then**
        $MigrateTo(n) \leftarrow MigrateTo(n) \cup \{i\}$
        $b_n \leftarrow x_n - m_{ij}$
      end if
    end for
  end for
    ▷ *Compute the number of migrated VNFs s*
  $s = \min(c_n, b_n)$
      ▷ *Migrate VNFs and update* $u_{n_{host}}$
  Migrate(s VNF type j, $MigrateTo(n)$, $u_{n_{host}}$)
end for

Figure 3.5 Algorithme MABS-Pull.

Cet algorithme vise à répartir les VNFs restant sans sauvegarde, après l'exécution de l'algorithme BS-Pull, entre les nœuds dont les VNFs disposent de sauvegardes afin de profiter de ces

sauvegardes existantes. En d'autres mots, MABS-Pull rapproche les VNFs sans sauvegarde vers les sauvegardes approvisionnées. En supposant que nous considérions d'abord le VNF de type $j \in J$, tous les nœuds de l'infrastructure physique sont supposés capables d'héberger des sauvegardes pour les VNFs de type j. Dans les étapes suivantes, notre objectif est de sélectionner le ou les nœuds pouvant réellement héberger les VNFs laissés sans sauvegardes et le nombre des VNFs par nœud.

Nous définissons, d'abord, les voisins $MigrateTo$ d'un nœud physique n ($MigrateTo(n)$) comme l'ensemble des nœuds physiques pouvant être atteints à partir de n dans une distance de d_{max} et que les nœuds de $MigrateTo(n)$ ont les ressources pour héberger plus de VNFs. Après avoir exécuté BS-Pull pour un certain type j, nous calculons l'ensemble des voisins $MigrateTo(n)$ pour chaque nœud physique $n \in N$ dont les VNFs ne possèdent aucune sauvegarde, et nous calculons également b_n, qui correspond au nombre de VNFs de type j qui devraient être migrés vers $MigrateTo(n)$ afin d'atteindre le nombre de sauvegardes déjà provisionnées par BS-Pull. Enfin, les VNFs sont migrés et l'ensemble du processus est répété pour tous les types de VNF.

3.4.5 Partage de sauvegarde optimisé (Optimized Backup Sharing - OBS)

Dans cet algorithme, nous migrons les VNFs avant d'allouer les sauvegardes. L'objectif est de minimiser le nombre de sauvegardes configurées par l'algorithme BS-Pull. La migration est effectuée après la première phase de BS-Pull où nous calculons à la fois les voisins $SNeigh(n)$ et le nombre de sauvegardes à allouer b_n.

La première phase de la migration consiste à déterminer les nœuds ayant encore des ressources parmi $SNeigh(n)$, qui seront référencés sous le nom $MigrateTo(n_{max})$, où n_{max} est le nœud avec le plus grand nombre de VNFs (c.-à-d. Le nombre de sauvegardes approvisionné). La deuxième phase consiste à distribuer une partie des VNFs intégrés à n_{max} parmi les nœuds de $MigrateTo(n_{max})$ afin d'obtenir un nombre équilibré de VNFs entre tous les nœuds. Les ressources de sauvegarde sont ensuite recalculées, allouées et associées à tous les voisins du nœud sélectionné.

Algorithm 5 OBS

Inputs
N: set of physical nodes
u_n: total number of VNFs hosted in physical node n
m_{ij}: number of type j VNFs hosted in physical node i
c_n: remaining capacity in each node $n \in N$
for $j \in J$ **do** ▷ *Parsing VNF types*
  $BNodes(j) \leftarrow$ ▷ *set of physical nodes whose type j VNFs have already backups*
  repeat
    Compute $SNeigh(n) \quad \forall n \in N$
      ▷ *Finding source neighbors of n*
    for $i \in SNeigh(n)$ **do**
      if $c_i \geq 0$ **then**
        $MigrateTo(n_{max}) \leftarrow MigrateTo(n_{max}) \cup \{i\}$
      end if
    end for
    ▷ *Compute the balanced number of VNFs s*
    $s = \frac{\sum m_{ij} / \ i \in MigrateTo(n_{max})}{|MigrateTo(n_{max})|}$
    ▷ *Migrate backups and update* $u_{n_{host}}$
    Migrate(s VNF type j, $MigrateTo(n)$, $u_{n_{host}}$)
    ▷ *Allocate backups and update* $u_{n_{host}}$
    Allocate (s backups, VNF type j, host n_{host})
    $BNodes(j) \leftarrow BNodes(j) \cup SNeigh(n_{host})$
  until $SNeigh(n) = \quad \forall n \in N$
  MABS-Pull ▷ *execute the MABS-Pull algorithm for a type j VNFs*
end for

Figure 3.6 Algorithme OBS.

Enfin, nous appliquons MABS-Pull pour maximiser la couverture des sauvegardes et l'ensemble du processus est appliqué à nouveau pour chacun des types de VNF.

3.5 Conclusion

Nous avons présenté, dans ce chapitre la formulation mathématique de notre problème sous forme d'un programme linéaire en nombre entier ayant comme objectif la réduction de nombre de sauvegardes en respectant un nombre de contraintes. Afin de résoudre le problème de disponibilité des ressources dans le cloud, nous avons proposé cinq algorithmes classés en deux catégories : BS-Pull, BS-Push et DBS les algorithmes qui n'utilisent pas la migration des VNFs et MABS-Pull et OBS les algorithmes qui l'utilisent.

Nous allons décrire l'environnement de nos simulations, discuter les résultats dans le but d'évaluer la performance de chacun de nos algorithmes ainsi que les résultats du programme linéaire en nombre entier.

CHAPITRE 4

EXPÉRIMENTATIONS ET RÉSULTATS

4.1 Introduction

Dans ce chapitre, nous évaluons les solutions proposées et nous les comparons avec la solution optimale fournie par CPLEX (IBM) en termes de nombre total de VNFs de sauvegarde, nombre de VNFs sans sauvegarde, temps d'exécution et le ratio de partage.

4.2 Expérimentation

4.2.1 Environnement de simulation

Pour évaluer les solutions proposées, nous avons implémenté les algorithmes en C et simulé l'infrastructure physique et l'intégration de la chaîne de services. Nous avons considéré un réseau avec 24 nœuds physiques avec des capacités de calcul différentes générées aléatoirement de 40 à 120 machines virtuelles. Pour simplifier, nous supposons que toutes les machines virtuelles ont les mêmes capacités (CPU, mémoire, disque) et qu'un VNF correspond à une seule machine virtuelle. De plus, les nœuds physiques sont connectés via 55 liens physiques générés aléatoirement. Nous supposons que l'approvisionnement des chaînes de services (c'est-à-dire les VNFs et les liens virtuels) est déjà effectué par l'algorithme de placement VNF existant. Dans nos expériences, nous avons utilisé l'algorithme d'allocation de ressources pour les chaînes de services proposé par Racheg *et al.* (2017).

4.2.2 Scénarios de simulation

Nous avons considéré 8 scénarios dans lesquels l'utilisation de l'infrastructure a été progressivement augmentée, comme le montre la Figure 4.1. Nous voyons sur la figure que nous avons des scénarios d'utilisation faible (S1 à S4) où l'utilisation est inférieure à 50% et également des

scénarios avec des utilisations élevées (par exemple, S5 à S11) où l'utilisation est supérieure à 50%.

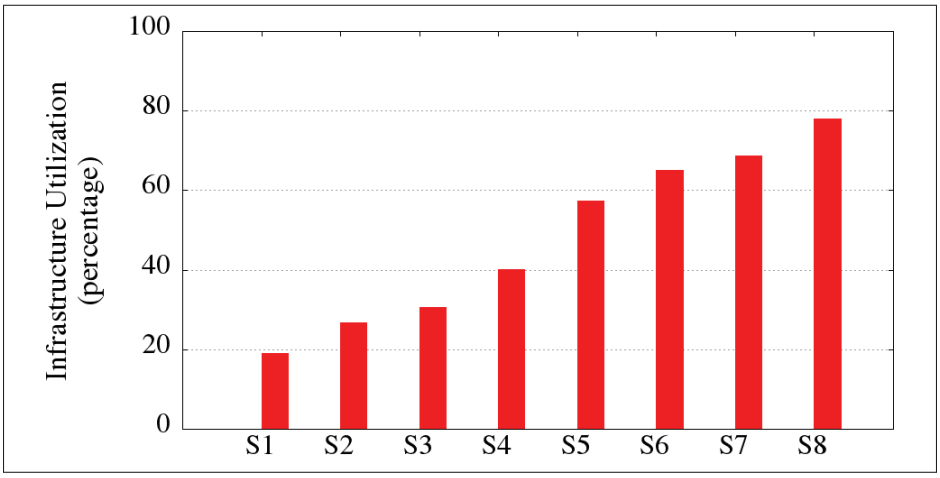


Figure 4.1 Les scénarios étudiés avec différents pourcentages d'utilisation d'infrastructure.

4.3 Résultats

Dans cette section, nous présentons les différents résultats des simulations effectuées pour évaluer la performance des solutions proposées.

4.3.1 Nombre de VNFs de sauvegarde

La Figure 4.2 compare le nombre total de sauvegardes obtenu avec les algorithmes heuristiques proposés à la solution optimale produite par CPLEX pour chaque scénario. Nous pouvons voir que pour les scénarios à faible utilisation (S1 à S4), les heuristiques fournissent un nombre légèrement supérieur de sauvegardes par rapport à la solution optimale fournie par CPLEX, ce qui indique que leurs solutions ne sont pas loin de la solution optimale. De plus, nous remarquons que DBS fournit généralement un nombre de sauvegardes inférieur à celui de PS-Pull / Push. Ceci est dû au fait que DBS distribue les sauvegardes entre plusieurs hôtes permettant le partage de ces sauvegardes parmi un plus grand nombre de VNFs ce qui diminue le nombre de sauvegardes.

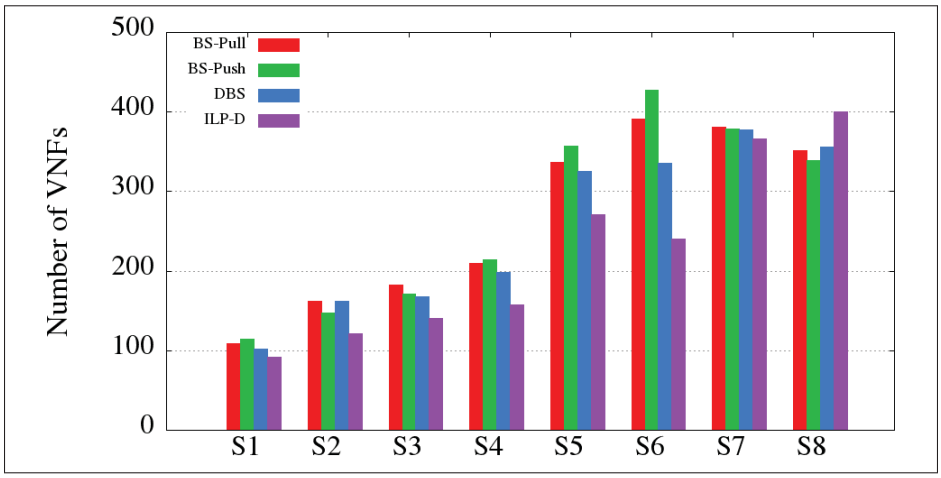


Figure 4.2 Nombre total de sauvegardes approvisionnées.

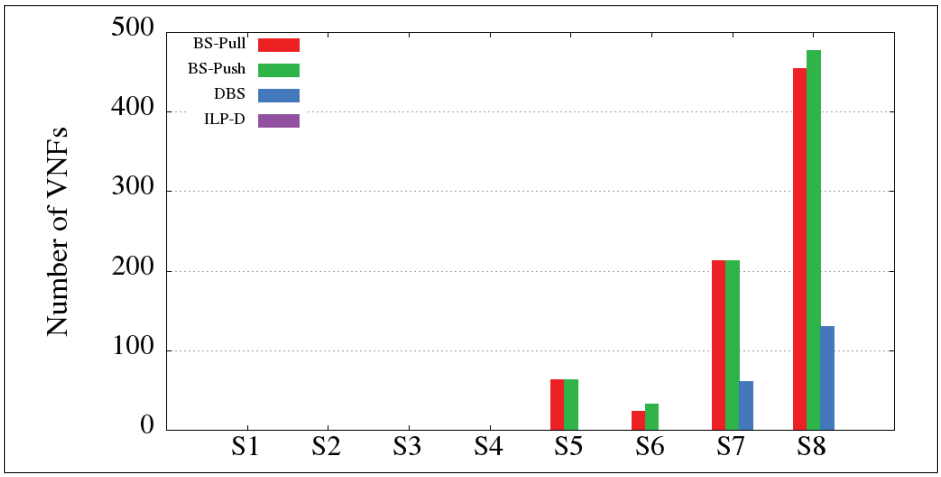


Figure 4.3 Nombre de VNFs sans sauvegarde.

Pour les scénarios à forte utilisation (S5 à S8), il devient plus difficile de trouver une solution réalisable, à savoir une solution garantissant que tous les VNFs dans l'infrastructure disposent de sauvegardes. Nous pouvons voir que, pour S5 et S6, seul CPLEX pourrait trouver une solution (ce qui est optimal) alors que les méthodes heuristiques ne garantissent pas l'approvisionnement de sauvegardes pour tous les VNFs, comme illustré à la figure 4.3. Cela montre que de nombreux VNFs restent sans sauvegardes en utilisant les heuristiques lorsque l'utilisation est élevée.

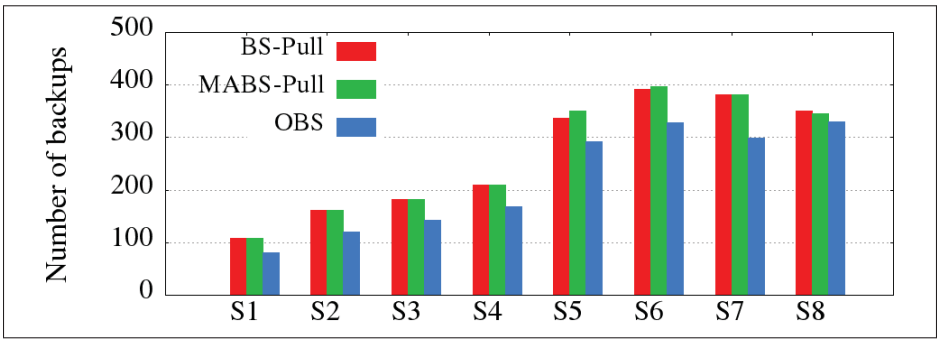


Figure 4.4 Nombre total de sauvegardes approvisionnées pour les algorithmes utilisant la migration.

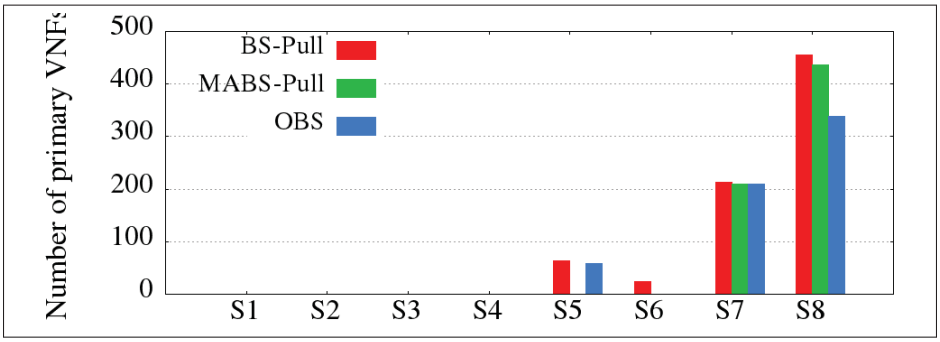


Figure 4.5 Nombre de VNFs sans sauvegarde pour les algorithmes utilisant la migration.

La figure 4.4 compare le nombre total de sauvegardes trouvées avec les deux algorithmes proposés et les algorithmes BS-Pull pour chacun des 8 scénarios. Pour les scénarios à faible utilisation (S1 à S4), le nombre de sauvegardes fournies par MABS-Pull est identique à celui fourni par BS-Pull. Il s'agit d'un résultat logique, car il ne reste aucun VNF sans sauvegarde après l'exécution de BS-Pull. Cependant, OBS fournit un nombre légèrement inférieur de sauvegardes par rapport à BS-Pull, car la migration dans OBS est effectuée avant l'attribution des sauvegardes dans le but d'optimiser le nombre de sauvegardes pour tous les scénarios testés.

Pour les scénarios à forte utilisation (S5 – S8), MABS-Pull fournit un nombre légèrement supérieur de sauvegardes, alors qu'il est toujours inférieur avec OBS, car il réduit le nombre de sauvegardes jusqu'à 20% (S6). En ce qui concerne les VNFs sans sauvegarde, MABS-Pull

et OBS réduisent leur nombre. MABS-Pull le réduit à zéro VNF pour S5 et S6 alors qu'il est encore considérablement élevé pour les deux derniers scénarios (S7, S8). Cependant, bien que le nombre de sauvegardes soit réduit, cette réduction dépend du scénario (c.-à-d. La manière dont les VNFs sont placés dans l'infrastructure).

4.3.2 Temps d'éxecution

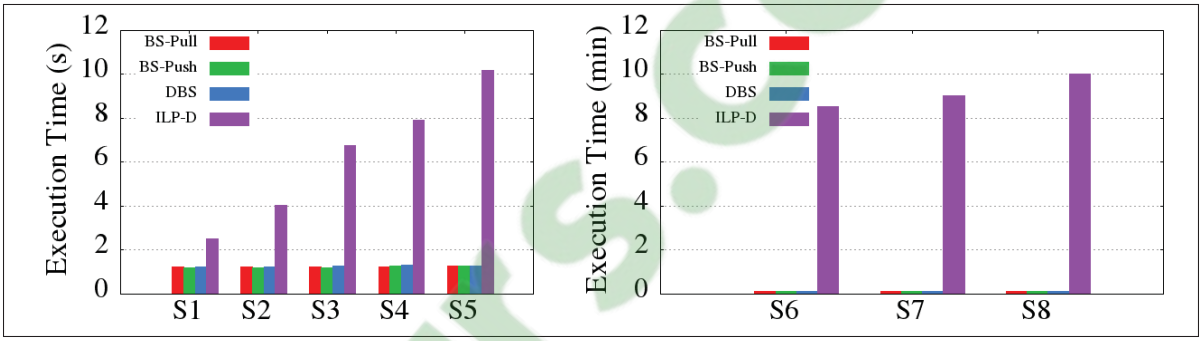


Figure 4.6 Temps d'exécution des différents algorithmes.

La figure 4.6 décrit le temps d'exécution des trois algorithmes proposés par rapport à celui de CPLEX pour chacun des 8 scénarios étudiés. Le temps d'exécution de CPLEX passe de 2 à 10 minutes (S6) à mesure que l'utilisation de l'infrastructure augmente. En effet, le nombre de variables dans le plan ILP augmente (par exemple, le nombre de nœuds et celui des VNFs) et rend le problème plus difficile à résoudre en raison du grand espace de recherche. Par contre, les temps d'exécution des autres algorithmes ne changent pas de manière significative à mesure que l'utilisation de l'infrastructure augmente.

La figure 4.7 illustre le temps d'exécution des deux algorithmes de migration proposés pour chacun des scénarios étudiés. La durée d'exécution des deux algorithmes est légèrement différente. OBS prend un peu plus de temps que MABS-Pull puisqu'il optimise le nombre de sauvegardes en répartissant le nombre de VNFs entre les nœuds avant d'allouer les sauvegardes et de migrer les VNFs restants après l'allocation.

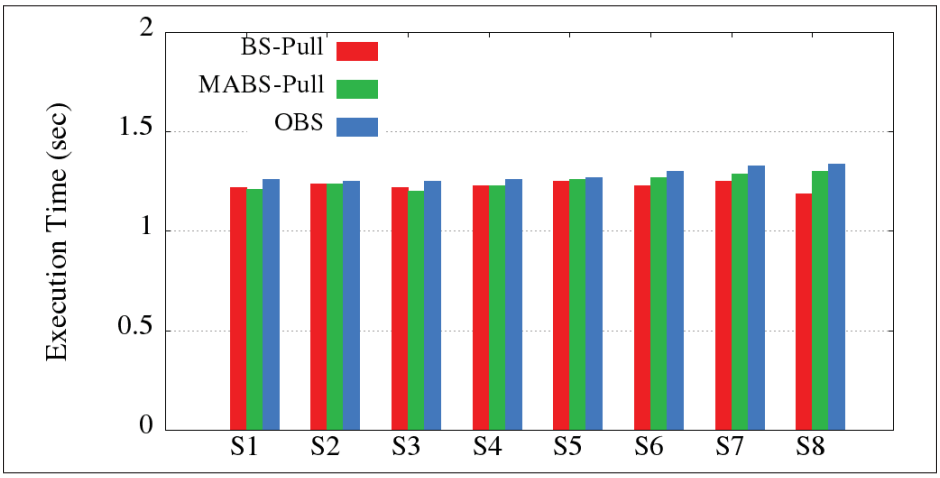


Figure 4.7 Temps d'exécution pour les algorithmes utilisant la migration.

4.3.3 Coût de synchronisation

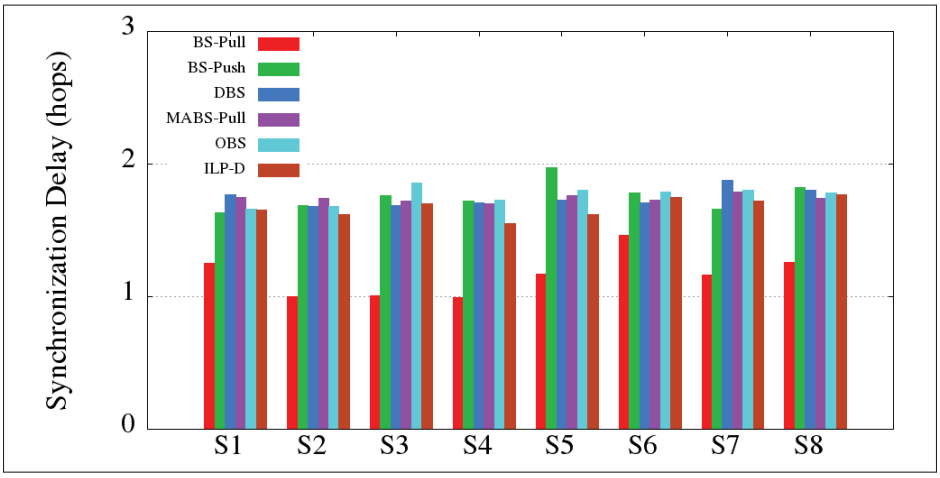


Figure 4.8 Le nombre de sauts moyen entre VNF et sa sauvegarde.

La figure 4.8 montre le nombre moyen de sauts entre un VNF intégré et sa sauvegarde pour les différentes solutions et selon les scénarios étudiés. Le nombre de sauts donne une idée du coût potentiel de la synchronisation en termes de délai et de bande passante. Les résultats montrent que pour les algorithmes, ainsi que CPLEX, le nombre de sauts est inférieur au nombre

maximal de sauts d_{max} spécifié en entrée de toutes les solutions (c.-à-d., d_{max} est égal à 2 dans nos expériences).

4.3.4 Ratio de partage

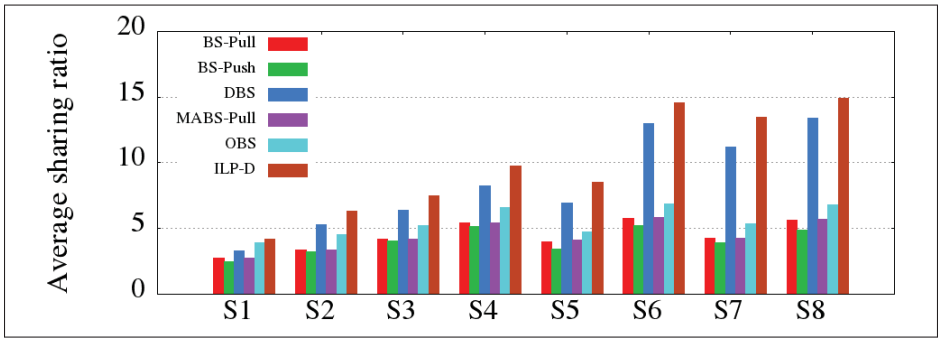


Figure 4.9 Le taux de partage moyen.

La figure 4.9 illustre le taux de partage moyen trouvé avec les trois algorithmes étudiés et pour tous les scénarios étudiés. Les résultats montrent que, quel que soit le scénario, BS-Pull fournit un ratio de partage légèrement supérieur à celui de BS-Push, tandis que la DBS fournit le ratio le plus élevé. Quant aux MABS-Pull et OBS, les résultats montrent qu'à partir de S5, nous pouvons remarquer une petite différence entre BS-Pull et MABS-Pull. Cependant, pour OBS, le ratio de partage est considérablement plus élevé que ces deux solutions, mais reste inférieur à celui de DBS.

4.4 Conclusion

Dans ce chapitre, nous avons commencé par présenter l'environnement expérimental puis discuter les résultats des simulations obtenus. En nous basant sur ces résultats, nous déduisons que, parmi les algorithmes qui n'utilisent pas la migration des VNFs, DBS comparé à BS-Pull et BS-Push permet de minimiser le nombre de sauvegardes ainsi que le nombre de VNFs sans sauvegardes. Cependant, en utilisant la migration, OBS optimise les résultats de BS-Pull pour les rapprocher de celle du DBS qui fournit toujours les meilleurs résultats.

CONCLUSION ET RECOMMANDATIONS

La réseautique définie par logiciel et la virtualisation des fonctions de réseau sont en train de changer la gestion des réseaux informatiques. En effet, les services informatiques sont offerts sous forme des chaînes de service composées par un ensemble des fonctions de réseau virtualisées. Ces chaînes de service traitent le trafic entrant et le dirigent vers sa destination. Malheureusement, dans de tels environnements, la défaillance d'un seul nœud peut entraîner la panne de plusieurs VNFs et, par conséquent, la rupture de nombreuses chaînes de services en même temps.

Des travaux récents ont abordé le problème de disponibilité des ressources des chaînes de service dans l'infonuage. Cependant, les solutions existantes peuvent être classées en deux catégories. Les techniques réactives ne pré-affectent pas des ressources pour la sauvegarde, mais traitent simplement des échecs quand ils se produisent. Cela entraîne un long temps de convergence après la défaillance, ce qui entraîne un temps d'arrêt du service plus long. D'autre part, les solutions proactives qui anticipent les défaillances et pré-allouent les ressources de sauvegarde afin d'assurer une restauration rapide du service en cas de défaillance.

Dans ce mémoire, nous avons proposé une nouvelle solution qui prévoit le nombre minimal de VNFs de sauvegarde partagée et minimise la quantité de ressources allouées à la sauvegarde. Nous avons, donc, formulé le problème comme un Programme Linéaire en Nombres Entiers puis proposé trois algorithmes heuristiques (BD-Pull, BD-Push et DBS) pour résoudre le problème dans des scénarios à grande échelle. Nos solutions visent à assurer la disponibilité des ressources à travers un ensemble de sauvegardes partagées entre les VNFs de même type indépendamment des chaînes de service auxquelles ils appartiennent. Nous avons, aussi, proposé deux autres algorithmes qui bénéficient de la migration des VNFs afin d'optimiser le rapport de partage en maximisant les VNFs ayant des sauvegardes ainsi qu'optimiser le nombre de sauvegardes.

Nous avons implémenté ces algorithmes en langage C et nous avons montré à travers un ensemble d'expérimentations, en considérant différents scénarios, que nos algorithmes fournissent des

solutions proches de la solution optimale fournie par CPLEX, tout en réduisant considérablement le temps d'exécution. Parmi les premiers trois algorithmes, DBS fournit le nombre minimum de sauvegardes. Par contre, après l'application des trois algorithmes, certains VNFs restent sans sauvegardes. Pour cela, nous avons proposé deux autres algorithmes OBS et MABS-Pull qui se basent sur la migration des VNFs. Les résultats montrent que nous avons réussi à diminuer le nombre de sauvegardes jusqu'à 20%.

Comme perspectives futures, nous visons à optimiser davantage les deux algorithmes afin de réduire leur complexité. Nous prévoyons également d'étendre ce travail pour prendre en compte les défaillances de plusieurs nœuds en même temps. Une autre voie intéressante serait l'utilisation de ces algorithmes pour le probléme d'approvisionnement axé sur la résilience des chaînes de service où la disponibilité des resources est considérée avant le placement et le chaînage des VNFs. Finalement, nous envisageons intégrer ces algorithmes comme des modules dans Openstack.

ANNEXE I

ON IMPROVING SERVICE CHAINS SURVIVABILITY THROUGH EFFICIENT BACKUP PROVISIONING

Nous présentons, dans cette première annexe, notre première publication faite dans le cadre de cette maitrise. Cet article nommé «On Improving Service Chains Survivability Through Efficient Backup Provisioning» a été présenté à la conférence «15th International Conference on Network and Service Management CNSM2018» à Rome (Italie) en novembre 2018. Dans cet article, nous avons introduit les deux algorithmes BS-Pull et BS-Push ainsi qu'une première version de la formulation mathématique (ILP) qui diffère de celle présentée précédemment dans ce rapport. La différence réside dans la façon dont le ILP alloue les sauvegardes aux nœuds physiques. En effet, dans cet article, ILP cherche à allouer les sauvegardes à un seul nœud physique.

14th International Conference on Network and Service Management (CNSM 2018)

On Improving Service Chains Survivability Through Efficient Backup Provisioning

Saifeddine Aidi*, Mohamed Faten Zhani*, Yehia Elkhatib*‡
*École de Technologie Supérieure (ÉTS Montreal), Montreal, Quebec, Canada
‡MetaLab, School of Computing and Communications, Lancaster University, UK
E-mail: saifeddine.aidi.1@ens.etsmtl.ca, mfzhani@etsmtl.ca, {i.lastname}@lancaster.ac.uk

***Abstract*—With the growing adoption of Software Defined Networking (SDN) and Network Function Virtualization (NFV), large-scale NFV infrastructure deployments are gaining momentum. Such infrastructures are home to thousands of network Service Function Chains (SFCs), each composed of a chain of virtual network functions (VNFs) that are processing incoming traffic flows. Unfortunately, in such environments, the failure of a single node may break down several VNFs and thereby breaking many service chains at the same time.**

In this paper, we address this particular problem and investigate possible solutions to ensure the survivability of the affected service chains by provisioning backup VNFs that can take over in case of failure. Specifically, we propose a survivability management framework to efficiently manage SFCs and the backup VNFs. We formulate the SFC survivability problem as an integer linear program that determines the minimum number of required backups to protect all the SFCs in the system and identifies their optimal placement in the infrastructure. We also propose two heuristic algorithms to cope with the large-scale instances of the problem. Through extensive simulations of different deployment scenarios, we show that these algorithms provide near-optimal solutions with minimal computation time.

I. Introduction

The emergence of Network Function Virtualization (NFV) and Software-Defined Networking (SDN) technologies is currently transforming the way networks are designed and managed as they provide operators much more flexibility to dynamically provision and configure network services. In particular, it is now possible to dynamically create chains of network services (Service Function Chains - SFCs) that can process incoming traffic and steer it across a chain of Virtual Network Functions (VNFs) like routers, IDSs and NATs that are running on virtual machines.

In the last few years, a large body of work has been dedicated to address resource provisioning and management of such SFCs [1]–[6]. Most of existing studies assume the complete availability of the physical infrastructure which is not realistic as failures are common in cloud network infrastructures [7]–[10]. Due to the dependency between virtual network functions in the chain, a single physical node failure in the network could easily bring down many VNFs and hence break several SFCs and make these services unavailable. Such downtime, even for few seconds, not only hurts the reputation of service providers but also incurs high revenue losses depending on the type of the offered service (e.g., $5,600 per minute according to [11]).

Existing proposals to manage failures and mitigate them can be broadly categorized into reactive and proactive techniques [12]. In proactive techniques, backup VNFs are provisioned whenever an SFC is received and embedded. These backups remain idle but are activated only when a failure occurs to take over the service and replace the failed VNFs [13]–[16]. The second category of existing solutions are reactive techniques. These techniques do not pre-allocate backup resources and deal with failures after they occur [15], [17], [18]. Consequently, they need additional time to allocate resources and provision new VNF instances to take over the service. This definitely results in a longer service disruption, which is very costly for service providers [12]. This makes proactive techniques more appealing even though they waste some resources for backup VNFs.

To remediate to this problem and minimize wastage of resources, several research efforts advocate to use shared backup VNFs [14] where the same backup resource can be used to mitigate the failure of a set of VNFs assuming that they do not fail at the same time (i.e., only a single VNF from this set can fail at a time). In this context, this paper investigates possible solutions to ensure the survivability of service chains against single physical node failures by using shared backup resources. Unlike previous work addressing the same problem where backups are shared only between the VNFs of the same chain [13] [14] , our solution assumes that backup VNFs are shared among all the chains embedded in the infrastructure. This significantly reduces the amount of resources used for the backup VNFs while still ensuring all SFCs are protected against single failures.

Our main goal is to ensure the survivability of all embedded SFCs against any single-node failure in the physical infrastructure. We reach this objective by proactively provisioning the minimal number backup VNFs to minimize resource wastage and by carefully placing them in the infrastructure. We also take into consideration the synchronization cost in terms of bandwidth and delay needed to keep the backup nodes up-to-date.

We can summarize the main contributions of this paper as follows:

- We propose a resource management framework with a survivability module. This module allows to provision and manage backup VNFs and it could be

978-3-903176-14-0

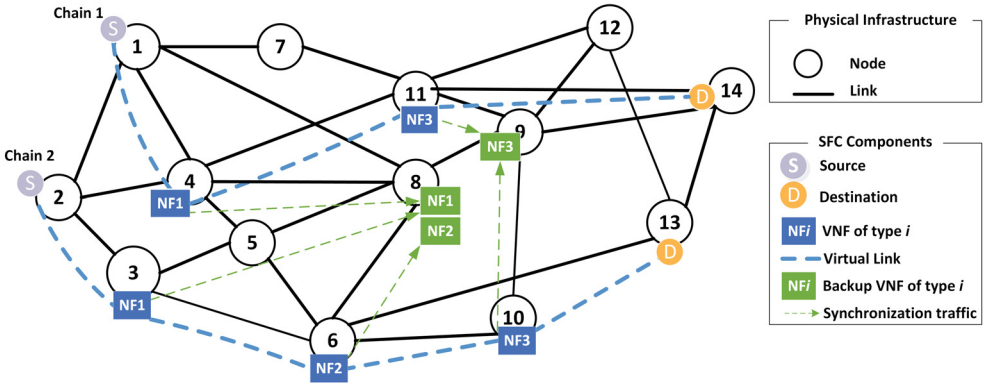


Figure 1: An example of various embedded SFCs sharing backups.

easily integrated into existing SFC resource management frameworks.

- We formulate the backup provisioning and placement problem as an Integer Linear Program (ILP) that finds the optimal number of shared backups for each type of VNF and optimally places them in the physical infrastructure.
- We devise two heuristic algorithms, called BS-Push and BS-Pull, respectively, that aim at solving the problem for large-scale scenarios within a reasonable timescale.
- We evaluate the performance of the proposed heuristics and compare it to the optimal solution found with the proposed ILP solved by the CPLEX optimizer.

The remainder of this paper is organized as follows. Section II provides a detailed description of service chain survivability problem. We discuss relevant related work in Section III. In Section IV and V, we mathematically formulate the addressed problem and then describe the proposed heuristic solutions. We present the experimental results in Section VI and follow up with some conclusions and future work in Section VII.

II. Problem Description

A Service Function Chain is made out from a set of different types of virtual network functions connected in a specific order to form a chain that steers the traffic from and to predefined source and destination [3]. A Virtual Network Function (VNF) is simply a virtual resource (i.e., virtual machine or container) that is running a specific network function (e.g., router, load balancer, NAT, IDS). To build the chain, the VNFs are connected through a set of virtual links having a sufficient amount of bandwidth to handle the traffic. Typically, service function chains are embedded into a physical infrastructure (referred to as NFV Infrastructure - NFVI [9]).

Figure 1 shows an example of two service chains mapped onto a wide area NFV infrastructure. The figure shows for each chain how the VNFs are embedded from the source to the destination. For instance, chain 2 has traffic coming from physical node 2 towards physical node 13 and it is composed of three VNFs of type 1, 2 and 3 that are embedded in physical nodes 3, 6 and 10, respectively. Chain 1 has only two VNFs of type 1 and 3 that are embedded in physical nodes 4 and 11, respectively.

Once SFCs are embedded into the NFVI, the operator faces the challenging task of ensuring the high survivability of these SFCs. In other words, they need to survive potential network failures in order to minimize service interruptions. However, as mentioned earlier, physical nodes are prone to failures and a single node failure may result in bringing down several VNFs and hence breaking multiple service chains. In this paper, we propose to ensure the survivability of the SFCs affected by a single failure by leveraging shared backups that could be used when the failure occurs. We also assume that a backup VNF should be of the same type of the set of VNFs it is backing up. In other words, a VNF of type i can only back up VNFs of type i. This assumption is reasonable as in practice the backup is a virtual machine that should implement a specific software and hence a backup VNF has to contain exactly the same software stack as the original VNF.

As an example of how shared backups could be placed, we can see in Figure 1 that physical node 9 hosts a backup of VNF type 3 (i.e., NF3) that is shared between the VNF type 3 of chain 1 and that of chain 2. If physical node 11 fails, and hence NF3 of chain 1 becomes out of service, the backup NF3 hosted in 9 takes over and replaces the failed function. Similarly, it can take over the service of NF3 belonging to chain 2 (hosted in node 10) if it fails. The figure also shows other examples of shared VNF backups (e.g., NF1 and NF2 hosted in node 8). It is easy to check that the two service chains shown in this example are perfectly survivable to any single node failure.

Furthermore, backup VNFs are continuously synchronized with the active VNFs to be ready to take over the service in case of failure (see green arrows in Figure 1). For instance,

the backup NF3 hosted in 9 has the state of NF3 hosted in physical node 11 and that of NF3 hosted in 10. Whenever a failure happens, the last state of the failed function will be used when the backup is activated. State synchronization can be done at a different level. For example, at the level of the virtual machine running the function (e.g., memory synchronization [19]) or using customized synchronization scripts depending on the type of the network function (e.g., synchronizing rules in firewalls).

To make sure that state synchronization is efficient, the latency between a VNF and its backup should not exceed a certain bound. Furthermore, synchronization of the VNF state may consume bandwidth that should be minimized. In our work, we ensure to minimize the synchronization delay and consumed bandwidth by limiting the number of hops between each VNF and its backup (for example, the number of hops is limited to 2 in Figure 1).

The main challenge that we are addressing in this paper is how to find the minimal number of backup nodes and to determine their optimal placement in the physical infrastructure for each type of VNF taking into account the synchronization delay and the cost in terms of bandwidth consumption.

III. Related Work

In this section, we provide an overview of representative work on the survivability problem. We note that most existing work focuses on virtual networks survivability and not SFCs. However, they are still valid for our case as a service function chain can be seen as a particular case of a virtual network with a specific topology. In the following, we summarize existing techniques to ensure the survivability of SFCs and virtual networks as either *reactive* or *proactive* [12].

Reactive techniques do not pre-allocate resources for backup but simply deal with a failure when it occurs. This leads to a long convergence time after the failure, resulting in a higher service downtime. On the other hand, proactive solutions anticipate failures and pre-allocate backup resources to ensure fast recovery of the service in case of failures.

Yu et al. [13] considered the case of a single-node failure and introduced two approaches to provision backup nodes. The first approach, called 1-redundant, redesigns the virtual network request into a survivable request by adding a single backup node. The second approach is called k-redundant where k is a constant that represents the number of backup nodes to be provisioned. The problem with these approaches is that a single redundant node may not be enough whereas k redundant nodes might be too much, and hence could lead to a wastage of resources. To address this limitation, the solutions presented in this current work aim at finding the optimal number of backup nodes when the proposed ILP is used or at least minimize it when the proposed heuristics are used.

In the same direction, Ayoubi et al. [14] explored the space between 1 and k to find the optimal number of backup nodes to be incorporated into the requested virtual network. However, in this solution, the backup virtual nodes are provisioned for each request and hence they are not shared with other virtual networks. Our work is different in that it provisions backups that are shared among all virtual nodes belonging to all virtual infrastructures (i.e., SFCs) embedded in the physical infrastructure. As a result, our solutions further reduce the total number of backups provisioned in the system.

Xiao et al. [17] proposed a topology-aware solution that ensures a rational resource allocation for the virtual network and a fail-over remapping based on a set of pre-computed detour-paths. Rahman et al. [15] also proposed a hybrid approach that benefits from a set of a possible backup detours for each link. These detours are proactively precomputed before the arrival of virtual network requests to allow fast re-routing in case of link failure.

Finally, Bo et al. [18] proposed a greedy algorithm that, in case of a link failure, searches for alternative resources to re-allocate the end-to-end path or re-embed the entire virtual network if resources are not sufficient. This may result in long convergence time and higher service downtime. In [21], Ayoubi et al. have demonstrated the NP-hard nature of the survivability-aware embedding and proposed a polynomial time heuristic algorithm to restore failed services while maintaining the QoS requirements in terms of delays in case of single-node failures. Multiple failures were addressed in [22] where a heuristic was introduced in order to find a backup node. The algorithm is based on filtering techniques to parse the solution space and to speed up the search process for backups.

We summarize in Table I the aforementioned solutions. The table presents the type of solution (i.e., reactive vs. proactive) and it indicates whether it is addressing a single or multiple failures, node or link failures and whether the backup are shared between the virtual nodes of all virtual networks or among the virtual nodes of a single virtual network. As shown in the Table, the novelty of our work lies in the idea of sharing backups between VNFs of the same type that belong to different or virtual networks (or service chains) rather than the same virtual network (or service chain), which further reduces the amount of backup resources while still ensuring the survivability of the virtual networks to any single failure.

IV. Survivability Management Framework

In this section, we propose a management framework that incorporates a survivability module. Figure 2 shows the main components of this framework. It is made out from the following modules:

- **Service Chain Provisioning Module:** This module allocates the resources for the service chains and instantiates the required virtual machines running the network functions. It also makes use of the SDN controller to provision the required amount of bandwidth and to set the required forwarding rules into the switches to steer the traffic across the VNFs composing each service chain.

Table I: Existing solutions vs. the proposed ones

Solutions	Type of solution		Single/Multiple failures		Node/Link failures		Support of Shared backups		
	Proactive	Reactive	Single	Multiple	Node	Link	Supported	Shared among all VNs	Shared among a single VN
Yu et al. [13]	x		x		x		x		x
Ayoubi et al. [14]	x		x		x		x		x
Guo et al. [20]	x		x		x			-	-
Xiao et al. [17]	x			x	x			-	-
Rahman et al. [15]	x		x			x		-	-
Bo et al. [18]		x	x		x			-	-
Ayoubi et al. [21]		x	x		x			-	-
Ghaleb et al. [22]		x		x	x	x		-	-
BS-Pull/BS-Push	x		x		x		x	x	

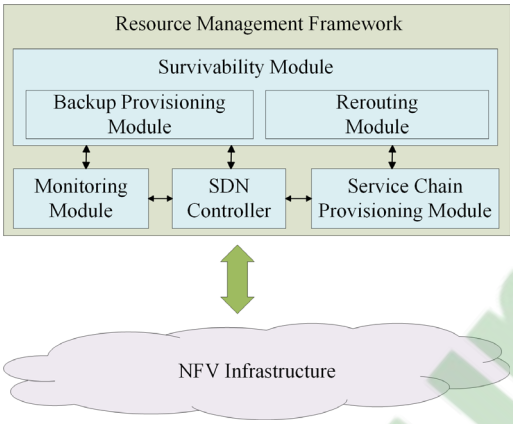


Figure 2: Architecture of the Proposed Resource Management Framework with the Survivability Module.

It is worth noting that the design of this module is out of the scope of this work. There is a large body of work addressing this module and any of the existing solutions could be used (e.g., [2], [4], [6]).

• **Monitoring Module:** This module is in charge of continuously monitoring the infrastructure physical nodes and links and of feeding other modules at real time with the state of the resources. When a failure is detected, the monitoring module reports to the survivability module, which, in turn, reacts to mitigate the failure and ensure service continuity.

• **Rerouting Module:** In case of failure, the rerouting module redirects the traffic, that is originally destined to the failed VNFs, to the backup VNFs.

• **Backup Provisioning Module:** This module is responsible for finding the minimal number of backups needed to ensure the survivability of the embedded chains and to determine their locations. This module also instantiates the backup VNFs and the synchronization links required to keep them up-to-date. In the following section, we describe in details the proposed solutions to provision backup VNFs while achieving the sought-after objectives.

V. Backup Provisioning Solutions

A. Integer Linear Program

In this section, we formulate the service chain survivability problem as an ILP aiming at minimizing the amount of resources allocated for the backup instances while ensuring a minimal synchronization cost and delay.

• **Infrastructure and chain modeling:** We model the physical infrastructure as a graph denoted by $G = (N, L)$ where N is the set of physical nodes and L is the set of physical links connecting them. Each physical node $n \in N$ has a computing capacity c_n. The capacity c_n is expressed as the maximal number of VNFs that can be hosted by physical node $n \in N$. For sake of simplicity, we assume that each VNF is running on a single virtual machine with a standard size. We hence can see c_n as the maximal number of virtual machines that could be provisioned in the physical node n. We also define d_{in} as the minimum number of hops separating the physical nodes i and n.

We model the service chain as a graph denoted by $S = (V, E)$ where V is the set of its composing VNFs and E is the set of virtual links connecting them. We assume there are different types of VNFs (e.g., Firewall, IDS, NAT). We denote by J the set of VNF types and we define m_{ij} as the number of VNFs of type $j \in J$ embedded in physical node i.

• **Decision Variables:** We define two decision variables. We denote by x_{ij} the number of backup VNFs of type j embedded in physical node i. We also define $y_{ijn} \in \{0,1\}$ to indicate whether the backups provisioned for the VNFs of type j embedded in the physical node i are hosted in the physical node n.

• **Problem constraints:** In order to find a feasible solution, several constraints must be satisfied. For instance, to ensure that the primary and backup VNFs are not

embedded in the same physical node, the following constraint must be satisfied:

$$y_{iji} = 0 \quad \forall i \in N, \forall j \in J \tag{1}$$

We also need to ensure that all VNFs of type j embedded in the physical node i necessarily have backups in another physical node:

$$\sum_{n \in N} y_{ijn} = 1 \quad \forall i \in N, \forall j \in J \tag{2}$$

Furthermore, if the backups for the type j VNFs embedded in the physical node i are hosted in physical node n then the total number VNF backups of type j provisioned in n should be higher or equal than the number of VNFs of type j embedded in node i. In other words, we have:

$$if \quad y_{ijn} = 1 \quad then \quad x_{nj} \geq m_{ij} \quad \forall i, n \in N, \forall j \in J \tag{3}$$

This if statement can be translated as the following constraint:

$$m_{ij} \leq x_{nj} + M(1 - y_{ijn}) \quad \forall i, n \in N, \forall j \in J \tag{4}$$

where M is a constant with a large value (in the vicinity of $10,000$).

Furthermore, to ensure that the physical node hosting the backups have sufficient resources, the following capacity constraint must be satisfied for every physical node n:

$$\sum_{j \in J} m_{nj} + \sum_{j \in J} x_{nj} \leq c_n \quad \forall n \in N \tag{5}$$

where the first term represents the number of VNFs hosted in the physical node n and the second term represents the number of VNF backups hosted in the same physical node.

Finally, as we have to minimize the synchronization cost and delay between the VNFs and their backups, we limit the number of hops between each VNF and its backup to a limited number of hops denoted by d_{max}. Thus, we have:

$$if \quad y_{ijn} = 1 \quad then \quad d_{in} \leq d_{max} \quad \forall i, n \in N, \ \forall j \in J \tag{6}$$

The previous statement can be also written as the following constraint:

$$d_{in} \leq d_{max} + M(1 - y_{ijn}) \quad \forall i, n \in N, \forall j \in J \tag{7}$$

where M is a constant with a large value.

It is also worth noting that, for sake of simplicity, we assume that the number of hops between two physical nodes reflects the time delay between them. However, this may not be always true. In this case, our model can be easily updated to consider the propagation delay between the nodes by defining d_{in} as the delay of the shortest path between nodes i and n, and d_{max} as the maximum delay required between a VNF and its backup.

• **Objective function:** Our ultimate goal is to minimize the amount of resources used by the backup VNFs while satisfying all the aforementioned constraints. This can be achieved by minimizing the total number of backups in all the physical nodes of the physical infrastructure. The objective function can be then written as:

$$\min \sum_{i \in N} \sum_{j \in J} x_{ij} \tag{8}$$

B. Heuristic Algorithms

In this section, we will present two heuristic solutions designed to solve the survivability problem. We call the first algorithm *Backup Sharing "Pull"* (BS-Pull) as we are looking at each physical node to find the maximum number of VNFs that it can backup (we refer to this as *pulling*). The second algorithm is called *Backup Sharing "Push"* (BS-Push) as the algorithm tries to push the coverage of the physical node in order to let it host backups of VNFs that are as spread as possible in several physical nodes.

Both algorithms are carried out in two phases. The first phase aims at finding the candidate physical nodes that satisfy the constraints of number of hops and the capacity for each type of VNFs. The second phase aims at selecting among

Algorithm 1 BS-Pull

Inputs
N: set of physical nodes
u_n: total number of VNFs hosted in physical node n
m_{ij}: number of type j VNFs hosted in physical node i
for $j \in J$ **do** ▷ *Parsing VNF types*
  $BNodes(j) \leftarrow \emptyset$ ▷ *set of physical nodes whose type j VNFs have already backups*
  repeat
    ▷ *Parsing all potential hosting nodes*
    for $n \in N$ **do**
      $SNeigh(n) \leftarrow \emptyset$ ▷ *set of source neighbors for physical node n*
      $b_n \leftarrow 0$ ▷ *number of type j VNFs able to use shared VNF backups hosted in node n*
      ▷ *Finding source neighbors of n*
      for $i \in N \backslash (BNodes(j) \cup \{n\})$ **do**
        if $d_{ni} \leq d_{max}$ & $m_{ij} + u_n \leq c_n$ **then**
          $SNeigh(n) \leftarrow SNeigh(n) \cup \{i\}$
          $b_n \leftarrow b_n + m_{ij}$
        end if
      end for
    end for
    ▷ *Finding the node n_{host} that maximizes the number of type j VNFs that are backed up*
    $n_{host} = \arg\max_n b_n$
    ▷ *Compute the number of shared backups s*
    $s = \max_{i \in SNeigh(n_{host})}(m_{ij})$
    ▷ *Allocate backups and update $u_{n_{host}}$*
    Allocate (s backups, VNF type j, host n_{host})
    $BNodes(j) \leftarrow BNodes(j) \cup SNeigh(n_{host})$
  until $SNeigh(n) = \emptyset \ \ \forall n \in N$
end for

the candidate nodes the ones that should host the backups. The difference between the two algorithms lies in the way the candidates hosting nodes are selected. In the following, we provide the details of the two proposed algorithms.

• **Algorithm BS-Pull:** Algorithm 1 describes the BS-Pull algorithm. It aims to allocate VNF backups for each VNF type one by one. Assuming we consider VNF type $j \in J$ first, all nodes in the physical infrastructure are assumed to be able to host backups for type j VNFs. Our goal in the following steps is to select which node or nodes could really host these backups and how many backups per node.

We first define *the source neighbors* of a physical node n (i.e., $SNeigh(n)$) as the set of physical nodes that could be reached from n within at most d_{max} hops and such as node n has enough resources to host the backup VNFs required to back up type j VNFs hosted in any of these source neighbors. In other words, if the backup VNFs are provisioned in node n, they can be shared among all the source neighbors of n.

For each physical node $n \in N$, we compute the set of source neighbors $SNeigh(n)$ and we also compute b_n, which is the number of VNFs of type j that could share the VNF backups that could be provisioned in physical node n (Lines 9-19). The higher b_n is, the higher is the number of VNFs sharing the backups. As a result, to maximize backup sharing, we select the node $n_{host} \in N$ that has the highest value of b_n to be the hosting node of the VNF backups. We then allocate the backup VNFs in the node n_{host} (function Allocate in Line 25). We repeat this operation until no source neighbors could be identified for all the physical nodes. Having no source neighbors for all physical nodes means that either there is no enough resources to host the backup VNFs (while satisfying the constraint on the number of hops) or there is no VNFs of type j that are left without backups. Finally, the whole process is repeated for all VNF types.

• **Algorithm BS-Push:** in this algorithm, we are adopting an approach different from the first one. For a particular physical node, our goal is maximize the number of its source neighbors that are using it (i.e., the physical node) to host their VNF backups (unlike BS-Pull that maximizes the number of VNFs that are backed up by the physical node but not the number of source neighbors using it).

As shown in Algorithm 2, similar to BS-Pull, BS-Push computes the set of source neighbors for all the physical nodes (Line 8). However, the algorithm selects the node n_{host} that has the highest number of source neighbors in order to host the VNF backups of all these neighbors (Line 10). The backup resources are then allocated and associated to all neighbors of the selected node (Line 14). The operation is then repeated until there are no more source neighbors for all physical nodes. Finally, the whole process is applied again for each of the VNF types.

Algorithm 2 BS-Push

1: **Inputs**
2: N: set of physical nodes
3: u_n: total number of VNFs hosted in physical node n
4: m_{ij}: number of type j VNFs hosted in physical node i
5: **for** $j \in J$ **do** ▷ *Parsing VNF types*
6: $BNodes(j) \leftarrow \emptyset$ ▷ *set of physical nodes whose type j VNFs have already backups*
7: **repeat**
8: Compute $SNeigh(n) \quad \forall n \in N$
9: ▷ *Finding the node n_{host} that is connected to to the maximum number of neighbors*
10: $n_{host} = \arg\max_n |SNeigh(n)|$
11: ▷ *Compute the number of shared backups s*
12: $s = \max_{i \in SNeigh(n_{host})}(m_{ij})$
13: ▷ *Allocate backups and update $u_{n_{host}}$*
14: Allocate (s backups, VNF type j, host n_{host})
15: $BNodes(j) \leftarrow BNodes(j) \cup SNeigh(n_{host})$
16: **until** $SNeigh(n) = \emptyset \quad \forall n \in N$
17: **end for**

VI. Simulation and Results

In this section, we compare the performance of the proposed algorithms with the optimal solution provided by CPLEX in terms of total number of backups and the execution time. To do so, we implemented the algorithms in C and simulated the physical infrastructure and the service chain embedding. We have considered a network with 24 physical nodes with different computing capacities randomly generated from 20 to 50 virtual machines. For simplicity, we assume that all virtual machines have the same resource capacities and that a single VNF is hoted by a single virtual machine. Furthermore, the physical nodes are connected through 55 physical links that were randomly generated. We assume the embedding of service chains (i.e., VNFs and virtual links) is already carried out by an existing VNF placement algorithm. In our experiments, we used the resource allocation algorithm for service chains that was proposed by Racheg et al. [4].

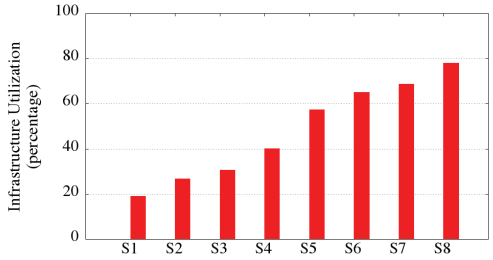


Figure 3: Studied scenarios with different infrastructure utilization.

We considered 8 different embedding scenarios where the utilization of the infrastructure has been gradually increased as shown in Figure 3. We can see in the figure

that we have low-utilization scenarios (i.e., s1 to s4) where utilization is less than 50% and also high-utilization scenarios (e.g., s5 to s8) where utilization is higher than 50%.

The objective of the experiments is to compare the number of backups and the number of VNFs left without backup provided by our algorithms with the one provided by CPLEX.

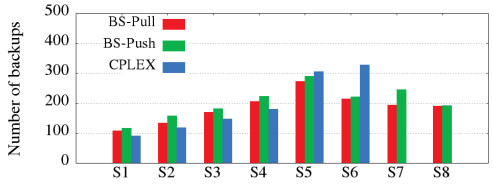


Figure 4: Total number of provisioned backups.

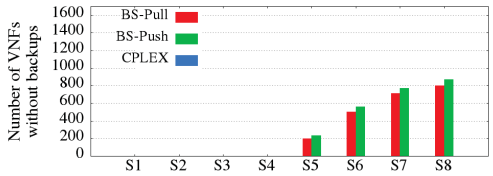


Figure 5: Number of VNFs without backup provisions.

A. Number of backups

Figure 4 compares the total number of backups found with the proposed heuristic algorithms with the optimal solution produced by CPLEX for each scenario. We can see that for low-utilization scenarios (i.e., S1–S4), the two heuristics provide a slightly higher number of backups compared to the optimal solution provided by CPLEX in low utilization scenarios, indicating that their solutions are not far from the optimal ones. In addition, we notice that BS-Pull generally provides lower numbers of backups compared to PS-Push.

For high-utilization scenarios (S5–S8), it becomes harder to find even a feasible solution, i.e., a solution that ensures that all VNFs in the infrastructures have backups. We can see that, for S5 and S6, only CPLEX could find a solution (which is optimal) whereas the two heuristics do not ensure that there are backups for all VNFs as depicted in Figure 5, which shows that many VNFs are left without backups using the heuristics when utilization is high.

Figure 4 also shows that for scenarios S7 and S8 that have high utilization (above 70%), CPLEX does not find an optimal solution that allow to provide backups for all VNFs. This is because there is no enough free resources in the infrastructure to provision all the required backups. The heuristics in this case still provide a solution even though several VNFs are left without backups as shown in Figure 5.

B. Execution Time

Figure 6 depicts the execution time of the two proposed algorithms compared to that of CPLEX for each of the 8 studied scenarios. The execution time for CPLEX goes from 2s to 7min (S6) as the infrastructure utilization increases. This is because the number of variables in the ILP increase (e.g., the number of nodes and that of VNFs) and makes the problem harder to solve because of the large research space. It is also clear from the figure that the two algorithms' execution times does not significantly change as the utilization of the infrastructure increases. We note, again, that beyond the sixth scenario (i.e., for S7–8), no optimal solution could be found due to the unavailability of free resources in the infrastructure.

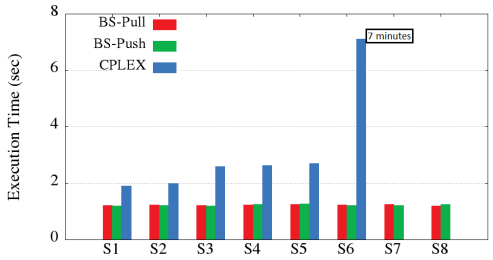


Figure 6: Execution time of the different algorithms.

C. Synchronization Cost

Figure 7 shows the average number of hops between an embedded VNF and its backup for the different solutions and across the studied scenarios. The number of hops provides some insight about the potential synchronization cost in terms of delay and bandwidth. The results show that for the two algorithms as well as CPLEX, the number of hops is below the maximal number of hops d_{max} specified as input to all solutions (i.e., d_{max} is equal to 2 in our experiments).

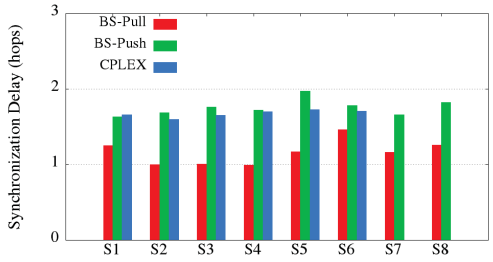


Figure 7: Synchronization distance of the different algorithms.

VII. Conclusion

In this paper, we addressed one of the uprising challenges faced by the infrastructure providers: the survivability of the service chains against node failures. We hence proposed a novel solution that provision the minimal number of shared backup VNFs that minimizes the amount of resources allocated for the backup.

We hence formulated the problem as an ILP and then proposed two heuristic algorithms to solve the problem for large-scale scenarios. Through extensive simulations, we demonstrated that our algorithms provide solutions that are close to the optimal one provided by CPLEX while they reduce the execution time considerably.

As a future work, we aim to further optimize both algorithms in order to reduce their complexity. We also plan to extend this work to take into consideration multiple node failures.

REFERENCES

[1] M. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 2, pp. 909–928, 2013.

[2] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 13, no. 3, pp. 518–532, 2016.

[3] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[4] W. Racheg, N. Ghrada, and M. F. Zhani, "Profit-driven resource provisioning in NFV-based environments," in *IEEE International Conference on Communications (ICC)*, 2017, pp. 1–7.

[5] L. Qu, C. Assi, and K. Shaban, "Network function virtualization scheduling with transmission delay optimization," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2016, pp. 638–644.

[6] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015.

[7] J. Lee, H. Ko, D. Suh, S. Jang, and S. Pack, "Overload and failure management in service function chaining," in *IEEE Conference on Network Softwarization (NetSoft)*, 2017, pp. 1–5.

[8] "Fail-slow at scale: When the cloud stops working," https://www.zdnet.com/article/how-clouds-fail-slow/, accessed: 2018-07-10.

[9] "ETSI GS NFV-REL 001 V1.1.1 (2015-01), Network Functions Virtualisation (NFV) Resiliency Requirements," https://goo.gl/PbQySQ, accessed: 2018-07-10.

[10] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable virtual data center embedding in clouds," *IEEE International Conference on Computer Communications (INFOCOM)*, April 27 - Mai 2 2014.

[11] "Downtime, outages and failures - understanding their true costs," https://www.evolven.com/blog/downtime-outages-and-failures-understanding-their-true-costs.html , accessed: 2018-07-13.

[12] M. F. Zhani and R. Boutaba, *Survivability and Fault Tolerance in the Cloud*. John Wiley & Sons, Inc, 2015, pp. 295–308. [Online]. Available: http://dx.doi.org/10.1002/9781119042655.ch12

[13] H. Yu, V. Anand, C. Qiao, and G. Sun, "Cost efficient design of survivable virtual infrastructure to recover from facility node failures," in *IEEE International Conference on Communications (ICC)*, 2011, pp. 1–6.

[14] S. Ayoubi, Y. Chen, and C. Assi, "Towards promoting backup-sharing in survivable virtual network design," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 3218–3231, 2016.

[15] M. R. Rahman and R. Boutaba, "SVNE: Survivable virtual network embedding algorithms for network virtualization," *IEEE Transactions on Network and Service Management*, vol. 10, no. 2, pp. 105–118, 2013.

[16] M. G. Rabbani, M. F. Zhani, and R. Boutaba, "On achieving high survivability in virtualized data centers," *IEICE Transactions on Communications*, vol. E97-B, no. 1, January 2014.

[17] A. Xiao, Y. Wang, L. Meng, X. Qiu, and W. Li, "Topology-aware virtual network embedding to survive multiple node failures," in *IEEE Global Communications Conference (GLOBECOM)*, 2014, pp. 1823–1828.

[18] L. Bo, T. Huang, X.-c. SUN, J.-y. CHEN, and Y.-j. LIU, "Dynamic recovery for survivable virtual network embedding," *The Journal of China Universities of Posts and Telecommunications*, vol. 21, no. 3, pp. 77–84, 2014.

[19] R. Boutaba, Q. Zhang, and M. F. Zhani, "Virtual machine migration: Benefits, challenges and approaches," in *Communication Infrastructures for Cloud Computing: Design and Applications*, H. T. Mouftah and B. Kantarci, Eds. USA: IGI-Global, 2013, pp. 383–408.

[20] B. Guo, C. Qiao, J. Wang, H. Yu, Y. Zuo, J. Li, Z. Chen, and Y. He, "Survivable virtual network design and embedding to survive a facility node failure," *IEEE Journal of Lightwave Technology*, vol. 32, no. 3, pp. 483–493, 2014.

[21] S. Ayoubi, C. Assi, L. Narayanan, and K. Shaban, "Optimal polynomial time algorithm for restoring multicast cloud services," *IEEE Communications Letters*, vol. 20, no. 8, pp. 1543–1546, 2016.

[22] A. M. Ghaleb, T. Khalifa, S. Ayoubi, K. B. Shaban, and C. Assi, "Surviving multiple failures in multicast virtual networks with virtual machines migration," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 899–912, 2016.

ANNEXE II

ON OPTIMIZING BACKUP SHARING THROUGH EFFICIENT VNF MIGRATION

Nous présentons, dans cette deuxième annexe, notre deuxième publication. Cet article nommé «On Optimizing Backup Sharing Through Efficient VNF Migration» a été publié et présenté au Workshop «IEEE PVE-SDN 2019» qui a eu lieu avec «5th IEEE International Conference on Network Softwarization NetSoft2019» à Paris (France) en juin 2019. Au sein de cet article, nous avons présenté les deux algorithmes (MABS-Pull et OBS) qui profitent de la migration des VNFs pour minimiser le nombre des VNFs laissés sans sauvegarde ainsi qu'optimiser le nombre de sauvegardes.

IEEE NetSoft 2019 - 3rd Workshop on Performance Issues in Virtualized Environments and Software Defined Networking (PVE-SDN)

On Optimizing Backup Sharing Through Efficient VNF Migration

Saifeddine Aidi*, Mohamed Faten Zhani*, Yehia Elkhatib*‡
*École de Technologie Supérieure (ÉTS Montreal), Montreal, Quebec, Canada
‡MetaLab, School of Computing and Communications, Lancaster University, UK
E-mail: saifeddine.aidi.1@ens.etsmtl.ca, mfzhani@etsmtl.ca, {i.lastname}@lancaster.ac.uk

***Abstract*—With the emergence of software defined networking and network function virtualization technologies, network services are expected to be offered as service function chains made out from virtual network functions that are connected to steer and process the incoming traffic. In this context, achieving the survivability of these chains against failures is a key challenge to ensure high availability and continuity of the services. A promising solution proposed in the literature is to provision backups for the virtual network functions that could be shared among multiple service chains. These backups are used in case of a failure to take over the failed functions and ensure service continuity.**

In this paper, we propose two solutions to efficiently place and provision the shared backups in order to ensure the survivability of the service chains against single node failures. The originality of these solutions is that they leverage the migration of virtual network functions to minimize the resources consumed by the backups. Simulation results show that, compared to existing solutions, the proposed schemes leveraging migration are able to reduce by up to 20% the amount of resources allocated for the shared backups while ensuring the survivability of the service chains .

I. Introduction

The emergence of Network Function Virtualization (NFV) and Software-Defined Networking (SDN) is changing the way networks are designed and managed by offering more flexibility to customize and configure network services. These services are hence provisioned and implemented as Service Function Chains (SFCs) that process and steer traffic through Virtual Network Functions (VNFs) towards the destination.

In this context, one key challenge is to ensure the survivability of service function chains against failures. Indeed, the failure of even one single physical node hosting several VNFs belonging to different service chains would bring down these chains and affect their offered services. It is known that failures are common in cloud infrastructures. For instance, major cloud providers like Amazon, Microsoft and IBM have suffered in 2017 from several outages that could last to up to 4 hours, which affects their reputation and may translate into hundreds of thousands of dollars of revenue loss [1].

Existing literature contains a large array of proposals to manage failures and mitigate their impact in order to ensure service chains' survivability [2]. Previous work proposed to proactively allocate backups for VNFs so that they can take over when a failure occurs [3], [4]. They also advocate to use backups that are shared among multiple VNFs belonging to different service chains in order to avoid wasting resources. As such, one shared VNF backup could be used to mitigate the failure of multiple VNFs assuming that they do not fail at the same time, i.e., only a single VNF from these VNFs is assumed to fail at a time.

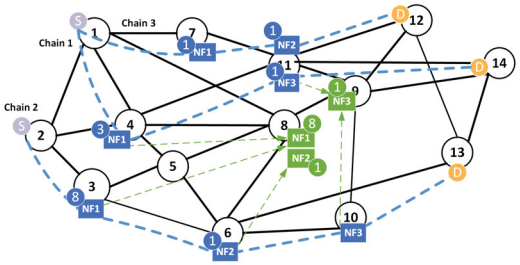
978-1-5386-9376-6/19/$31.00 ©2019 Crown

In this paper, we further investigate this solution. However, unlike previous work [2]–[4], we propose novel solutions that (1) decide on the number and the placement of the shared backups and (2) leverage VNF migration (i.e., the migration of the virtual machine or container hosting the network function) to relocate the VNFs in order to minimize the number of shared VNF backups.

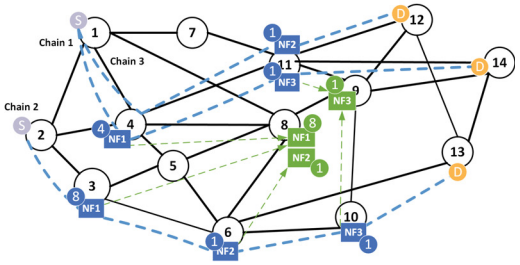
We can summarize the main contribution of this paper as follows:

- We devise a first backup provisioning scheme, called MABS-Pull, that starts by deciding of the number of shared backups and their placements and then uses migration to relocate the VNFs to ensure that all VNFs have backups.
- We devise a second backup provisioning scheme, called OBS, that leverages migration to relocate VNFs before allocating the shared backups and also after their allocation, which allows to further optimize the results.
- Through extensive simulations, we evaluate the performance of the two proposed schemes and compare them to an existing solution [3] that does not leverage VNF migration. Simulations show that OBS could reduce up to 20% the amount of resources allocated to shared backups while providing the same level of survivability achieved by the existing solution.

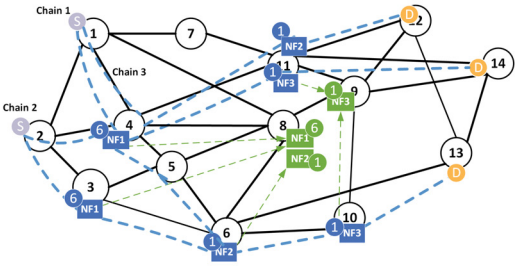
The remainder of this paper is organized as follows. Section II provides a detailed description of the service chain survivability problem and showcases how migration could be leveraged to reduce the amount of resources allocated for shared backups. We then discuss relevant related work in Section III. In Section IV, we present the proposed heuristic solutions. We describe the experimental results in Section VI. Finally, we present some conclusions and describe potential future work in Section VII.



(a) Original mapping of the service chain VNFs and their backups: the instance of NF1 hosted in 7 has no backup



(b) Mapping after the migration of the VNF NF1 from node 7 to 4: all instances of NF1 have now backups



(c) Mapping after the migration of 2 instances of NF1 from node 3 to 4: the number of needed backups for NF1 drops from 8 to 6

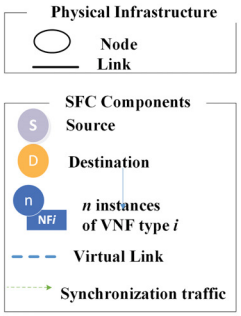


Figure 1: Example showing how VNF migration could allow (1) to ensure to have backups for all VNFs and (2) to reduce the number of backups

II. Problem Description

A service function chain is a set of different types of VNFs (e.g., router, load balancer, NAT, IDS). These VNFs are connected through a set of virtual links in a specific order to form a chain to steer the traffic from the source to the destination [5]. Service function chains are embedded into a physical infrastructure referred to as NFV Infrastructure (NFVI) [6].

Figure 1 shows an example of three service chains mapped onto a wide area NFV infrastructure and it shows also the backups that are provisioned to ensure the survivability against any single failure. The figure shows how the VNFs are placed from the source to the destination for each chain. For instance, chain 2 has traffic coming from physical node (2) towards physical node (13) and it is composed of three VNFs of type NF1, NF2 and NF3 that are embedded in physical nodes (3), (6) and (10), respectively. As an example of placing the shared backups, we can see in Figure 1 (a) that physical node (8) hosts 8 backup instances of VNF type NF1 that are shared between the 3 VNFs (type NF1) of chain 1 and the 8 VNFs (type NF1) of chain 2. If physical node (4) fails, the 3 VNFs (type NF1) of chain 1 become out of service, 3 backups type NF1 from those hosted in (8) take over and replace the failed functions. It is also easy to see that any single node failure could be mitigated using the provisioned backups. It is also worth noting that the VNF backups need to be continuously synchronized with the corresponding primary VNFs to save the last state of the failed function – see green arrows in Figure 1 (a).

Some VNF instances may be left without backups due to the lack of resources to provision backups. For instance, we can see in Figure 1 (a) that there is an instance of VNF type NF1 hosted in node (7) that has no backup. Figure 1 (b) shows that VNF migration could be leveraged to relocate the VNF that has no backups to other physical nodes where they can benefit from existing shared backups. The figure shows that the VNF type NF1 of chain 3 hosted in node (7) has been migrated to node (4), and thus, it is now backed up by one of the 8 backups already embedded in node (8).

Migration could be also leveraged to reduce the number of backups. This can be done if we distribute the VNFs fairly among the nodes that are close to the one hosting backups. Figure 1 (c) shows how relocating the VNFs could allow to

reduce the number of needed backups. In the figure, two VNFs type NF1 of chain 1 embedded in node (3) were migrated to node (4) and hence, with this new configuration, only 6 backups are needed rather than 8.

It worth noting that, in any of the three configurations (a), (b) and (c) illustrated in Figure 1, any single node failure could be mitigated using the provisioned backups.

We also note that migration has a cost (e.g., in terms of bandwidth and cpu consumption and service interruption as well [7]). To ensure a minimal migration cost, we assume in this work that a migration could be carried out only if the number of hops between the original location of a VNF and the new one does not exceed a maximal number of hops. For instance, the number of hops is limited to 3 in the example provided in Figure 1.

To conclude, as illustrated in the aforementioned example, the main challenge addressed in this paper is to leverage migration to ensure that all VNFs have backups and also to minimize the number of these backups by efficiently relocating the VNFs.

III. Related Work

In this section, we provide an overview of related work on the survivability problem. We also present some relevant work on VNF migration as our proposal relies on migration.

A. Backup provisioning

Reactive techniques do not pre-allocate backup resources but simply deal with a failure when it occurs which leads to a longer interruption time. As this work focuses on proactive techniques, we will focus mainly on proactive solutions.

Yu et al. [8] introduced two approaches to provision backup nodes to address a single-node failure. Both approaches redesign the virtual network request into a survivable network by adding backup nodes. The difference between the two approaches is the number of nodes added. The first approach, called 1-redundant, adds a single backup node whereas the second approach, called k-redundant, adds k backups nodes. The main limitation of this approach is the wastage of resources as the number of backups k is constant.

To address this limitation, Ayoubi et al. [4] aimed to find the optimal number of backup nodes to be added to the requested virtual network by exploring the space between 1 and k.

To further optimize the number of backups, we proposed in our previous work [3] two schemes, called BS-Pull and BS-Push, that provision backups that are shared among VNFs belonging to different service chains. These schemes are able to protect the chains again single node failures and significantly reduce the total number of backups provisioned in the system.

As all the aforementioned solutions do not leverage VNF migration to further optimize the placement and the number of backups requested to ensure the survivability of the service chains, this current work focuses on migration and shows how it can further reduce the number of the shared backups while achieving the survivability of the chains.

B. VNF Migration

Nobach et al. [9] propose SliM, a statelet-based framework for seamless VNF state migration that are based on a novel interface added to the VNF, sends statelets of the snapshot of the source VNF instance over the corresponding stream to the destination instance. This VNF state is responsible for replicating status of VNF instances of the same type in the service chain. Compared to the duplication-based model, that incur significant high additional costs according to them, SliM performs 3 times better in terms of link utilization. However, the authors did not take into consideration the distance between the VNF source and its destinations and the cost of the communication raised due to congested links.

Peuster and Karl [10] proposed E-State, a management framework that automatically handles the migration of the state of the VNF. The framework shares logically the VNF state by creating distributed state memory. Compared to centralized management system, E-State provide a better performance in terms of the number of replicated instances. One limitation of this work is that the authors do not take into consideration a service chain composed of multiple instances.

As the solutions proposed in this paper are based on our previous work [3], we provide in the following a brief overview of the the framework proposed in [3].

IV. Survivability-aware Management Framework

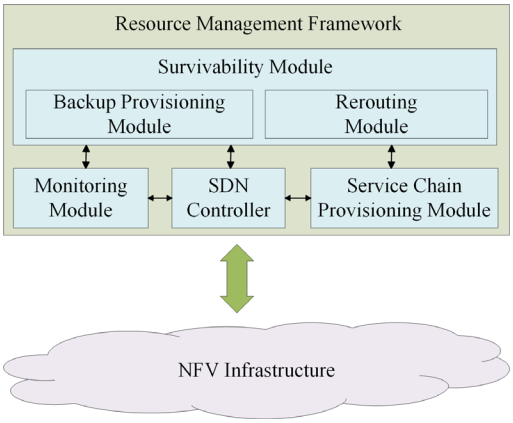


Figure 2: Architecture of the Proposed Resource Management Framework with the Survivability Module [3].

Figure 2 shows the survivability-aware service chain management framework proposed in [3]. In additional to traditional modules like the service chain provisioning module, the monitoring module and the SDN controller, we advocate to have a survivability module. This module is responsible for finding the minimal number of needed backups and for deciding where they should be provisioned (i.e., their locations) in order to protect the service chains against single node failures.

The module is implemented using the BS-Pull algorithm [3]. The BS-Pull algorithm identifies the number of shared backups for each types of VNF and decide of their placement in the physical infrastructures. It also minimize the synchronization cost and delay between the VNF and its backup by satisfying the two hop constraint, that is to ensure that the backup of a particular VNF is only two hops away from this VNF.

Unfortunately, with BS-Pull, some VNFs may remain without backups because of the lack of free resources to provision these backups or the inability to satisfy the two hop constraint. The solutions proposed in this paper aim at addressing this limitation by leveraging VNF migration in order to ensure that all VNFs have backups and that their placement satisfies the two hop constraint.

V. Migration-Aware Backup Provisioning

Before providing the details of the proposed schemes, we define a new metric called *the sharing ratio* as the total number of VNFs divided by the total number of their backups. If this ratio is high, it means that, on average, a high number of VNFs have a small number of shared backups. In other words, the higher is this ratio, the better is the provisioning scheme.

The objective of the migration in both proposed schemes is to maximize the sharing ratio by migrating the VNFs that are left with no backups. The first proposed scheme, called MABS-Pull (i.e., Migration After BS-Pull), migrates the VNFs after running the BS-Pull algorithm. The second proposed scheme is called OBS (i.e., Optimized Backup Sharing) and uses migration before and after applying the BS-Pull algorithm in order to optimize the number of backups to be allocated.

Both schemes are carried out in two phases. The first phase aims at finding the candidate physical nodes that satisfy the constraints of number of hops and the capacity. The second phase aims at migrating a certain number of VNFs to each of the candidate nodes. The difference between the two algorithms lies in the timing at which we execute the migration, as well as the objective of the migration itself. In the following, we provide the details of the two proposed schemes.

• **Algorithm MABS-Pull:** The scheme aims to distribute the VNFs that have no backups among the nodes whose VNFs have backups to further benefit from these existing backups. Assuming we consider VNF type $j \in J$ first, all nodes in the physical infrastructure are assumed to be able to host backups for type j VNFs. Our goal in the following steps is to select which node or nodes could really host the VNFs left without backups and how many VNFs per node.

We first define *the MigrateTo neighbors* of a physical node n (i.e., $MigrateTo(n)$) as the set of physical nodes that could be reached from n within at most d_{max} hops and such as the nodes in $MigrateTo(n)$ have some resources to host more VNFs. After executing BS-Pull for a certain type j, we compute the set of MigrateTo neighbors $MigrateTo(n)$ for each physical node $n \in N$ whose VNFs have no backups, and we also compute b_n, which is the number of VNFs of type j that should be migrated to $MigrateTo(n)$ in order to reach the number of backups already provisioned by BS-Pull. Finally, the VNFs are migrated and the whole process is repeated for all VNF types.

Algorithm 1 MABS-Pull

Inputs: Output of BS-Pull for each VNF type
N: set of physical nodes
u_n: total number of VNFs hosted in physical node n
m_{ij}: number of type j VNFs hosted in physical node i
$BNodes(j)$: set of physical nodes whose type j VNFs have already backups
$BHosts(j)$: set of physical nodes hosting type j backups
$x_{n_{host}}$: number of type j backups hosted in the node n_{host}
c_n: remaining capacity in each node $n \in N$
for $j \in J$ **do** ▷ *Parsing VNF types*
  for $n \in N$ **do**
    $MigrateTo(n) \leftarrow \emptyset$ ▷ *set of candidate nodes to host migrated VNFs from a physical node n*
    $b_n \leftarrow 0$ ▷ *number of type j VNFs to be migrated*
    ▷ *Finding candidate nodes to host VNFs migrated from n*
    for $i \in BNodes(j) \backslash (BHosts(j) \cup \{n\})$ **do**
      if $d_{ni} \leq d_{max}$ & $c_n \geq 0$ **then**
        $MigrateTo(n) \leftarrow MigrateTo(n) \cup \{i\}$
        $b_n \leftarrow x_{n_{host}} - m_{ij}$
      end if
    end for
  end for
  ▷ *Compute the number of migrated VNFs s*
  $s = \min(c_n, b_n)$
  ▷ *Migrate VNFs and update $u_{n_{host}}$*
  Migrate(s VNF type j, $MigrateTo(n)$, $u_{n_{host}}$)
end for

• **Algorithm OBS:** In this scheme, we are actually migrating the VNFs before allocating the backups. The goal is to minimize the number of backups provisioned. The migration is done after the first phase of the BS-Pull where we compute both the neighbors $SNeigh(n)$ and the number of backups to allocate.

The first phase of the migration is to determine the nodes having some resources left among $SNeigh(n)$ which will be refereed to as $MigrateTo(n_{max})$ where n_{max} is the node with the highest number of VNFs (i.e. the number of backups provisioned). The second phase consist in distributing some of the VNFs embedded in n_{max} among the nodes in $MigrateTo(n_{max})$ to get a balanced number of VNFs among all nodes. The backup resources are then recomputed, allocated and associated to all neighbors of the selected node.

Finally, we apply the MABS-Pull to maximize the coverage of the backups and the whole process is applied again for each

of the VNF types.

Algorithm 2 OBS

Inputs
N: set of physical nodes
u_n: total number of VNFs hosted in physical node n
m_{ij}: number of type j VNFs hosted in physical node i
c_n: remaining capacity in each node $n \in N$
for $j \in J$ **do** ▷ *Parsing VNF types*
  $BNodes(j) \leftarrow \emptyset$ ▷ *set of physical nodes whose type j VNFs have already backups*
  repeat
    Compute $SNeigh(n) \quad \forall n \in N$
      ▷ *Finding source neighbors of n*
    for $i \in SNeigh(n)$ **do**
      if $c_i \geq 0$ **then**
        $MigrateTo(n_{max}) \leftarrow MigrateTo(n_{max}) \cup \{i\}$
      end if
    end for
    ▷ *Compute the balanced number of VNFs s*
    $s = \frac{\sum m_{ij} / \ i \in MigrateTo(n_{max})}{|MigrateTo(n_{max})|}$
    ▷ *Migrate backups and update $u_{n_{host}}$*
    Migrate(s VNF type j, $MigrateTo(n)$, $u_{n_{host}}$)
    ▷ *Allocate backups and update $u_{n_{host}}$*
    Allocate (s backups, VNF type j, host n_{host})
    $BNodes(j) \leftarrow BNodes(j) \cup SNeigh(n_{host})$
  until $SNeigh(n) = \emptyset \quad \forall n \in N$
  MABS-Pull ▷ *execute the MABS-Pull algorithm for type j VNFs*
end for

VI. Simulation and Results

In this section, we compare the performance of the proposed backup provisioning schemes with the performance of the BS-Pull algorithm that does not use migration and that was proposed in [3]. The comparison is done in terms of total number of backups and the execution time and sharing ratio. To do so, the proposed algorithms were implemented in C. We simulated a physical network having 55 physical links and 24 physical nodes. These nodes have different hosting capacities ranging from 40 to 120 VNF instances.

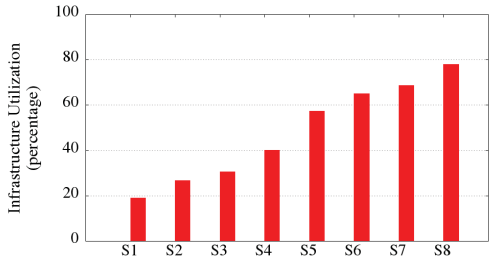


Figure 3: Infrastructure utilization for the studied scenarios.

We considered 8 different embedding scenarios provided by an existing VNF placement algorithm proposed by Racheg et al. [11] where the utilization of the infrastructure has been gradually increased. Figure 3 shows that we have low-utilization scenarios (i.e., S1–S4) where utilization is less than 50% and high-utilization scenarios (i.e., S5–S8) where utilization goes from 50% up to almost 80%.

A. Number of backups

Figure 4 compares the total number of backups found with the two proposed schemes compared to the original BS-Pull algorithm for each of the 8 scenarios. For low-utilization scenarios (i.e., S1 to S4), the number of backups provided by MABS-Pull is the same as the one provided by BS-Pull which is a normal as there is no VNFs left without backups after executing BS-Pull. However, OBS provides a slightly fewer number of backups compared to BS-Pull because the migration in OBS is carried out before the allocation of the backups aiming to optimize the number of backups for all tested scenarios.

For high-utilization scenarios (i.e., S5–S8), MABS-Pull provides a slightly higher number of backups whereas it is always less with OBS, which reduces by up to 20% the number of shared backups (e.g., scenario S6).

As to the number of VNFs left without backups (Figure 5), both MABS-Pull and OBS significantly reduce this number. We can see that there are VNFs left without backups for MABS-Pull for scenarios S5 and S6 while there are many for the last two scenarios S7 and S8. This is mainly due to the lack of free resource in the infrastructure to provision backups.

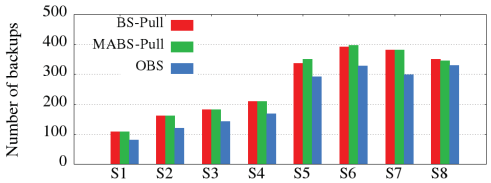


Figure 4: Total number of provisioned backups.

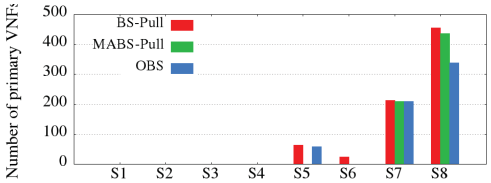


Figure 5: Number of VNFs without backup.

B. Execution Time

Figure 6 depicts the execution time of the two proposed migration schemes for each of the studied scenarios. The execution time of both schemes are slightly

different. OBS takes slightly more time than MABS-Pull as it is optimizing the number of backups by distributing the number of VNFs between the nodes before allocating the backups and migrating the left VNFs after the allocation.

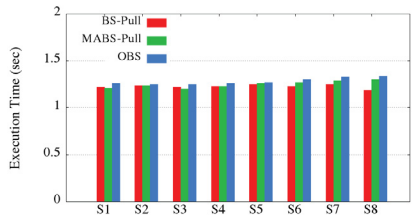


Figure 6: Execution time of the proposed solutions.

C. Infrastructure Utilization

Figure 7 shows the infrastructure utilization after the execution of the three schemes. Since MABS-Pull is providing the same number of backups as BS-Pull for low utilization scenarios (i.e., S1–S4), they both result in the same resource utilization. However, OBS leads to lower utilization as it optimizes the number of backups for all the studied scenarios. We can also see that, for high-utilization scenarios, the utilization remains high for the three algorithms, although OBS results in a slightly reduced utilization.

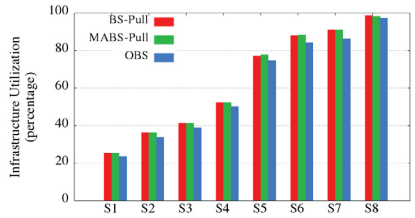


Figure 7: Infrastructure utilization after applying the different algorithms.

D. Sharing Ratio

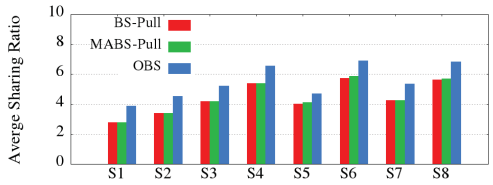


Figure 8: Average sharing ratio.

Figure 8 depicts the average sharing ratio found with the three studied schemes and for all the studied scenarios. The results show that starting from S5, we can notice a small difference between BS-Pull and MABS-Pull. However, for OBS, the sharing ratio is considerably higher than both solutions. These results demonstrate that using MABS-Pull and OBS allows to maximize the sharing ratio.

VII. Conclusion

Ensuring the survivability of service function chains is a challenging task for cloud providers. In this paper, we proposed two solutions to ensure the survivability of the service chains against single-node failures by proactively provisioning backups for the VNFs composing the chains. The originality of the proposed schemes is that they leverage VNF migration to reduce the number of the provisioned backups and to further optimize their placement. The conducted simulations showed that they are able to reduce by up to 20% the amount of resources allocated to VNF backups.

References

[1] "The 10 biggest cloud outages of 2017," https://www.crn.com/slide-shows/cloud/300089786/the-10-biggest-cloud-outages-of-2017-so-far.htm, accessed: 2018-07-10.

[2] M. F. Zhani and R. Boutaba, *Survivability and Fault Tolerance in the Cloud*. John Wiley & Sons, Inc, 2015, pp. 295–308. [Online]. Available: http://dx.doi.org/10.1002/9781119042655.ch12

[3] S. Aidi, M. F. Zhani, and Y. Elkhatib, "On improving service chains survivability through efficient backup provisioning," in *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE, 2018, pp. 108–115.

[4] S. Ayoubi, Y. Chen, and C. Assi, "Towards promoting backup-sharing in survivable virtual network design," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 3218–3231, 2016.

[5] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[6] "ETSI GS NFV-REL 001 V1.1.1 (2015-01), Network Functions Virtualisation (NFV) Resiliency Requirements," https://goo.gl/PbQySQ, accessed: 2018-07-10.

[7] R. Boutaba, Q. Zhang, and M. F. Zhani, "Virtual machine migration: Benefits, challenges and approaches," in *Communication Infrastructures for Cloud Computing: Design and Applications*, H. T. Mouftah and B. Kantarci, Eds. USA: IGI-Global, 2013, pp. 383–408.

[8] H. Yu, V. Anand, C. Qiao, and G. Sun, "Cost efficient design of survivable virtual infrastructure to recover from facility node failures," in *IEEE International Conference on Communications (ICC)*, 2011, pp. 1–6.

[9] M. Peuster and H. Karl, "E-state: Distributed state management in elastic network function deployments," in *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*. IEEE, 2016, pp. 6–10.

[10] L. Nobach, I. Rimac, V. Hilt, and D. Hausheer, "Slim: Enabling efficient, seamless nfv state migration," in *Network Protocols (ICNP), 2016 IEEE 24th International Conference on*. IEEE, 2016, pp. 1–2.

[11] W. Racheg, N. Ghrada, and M. F. Zhani, "Profit-driven resource provisioning in NFV-based environments," in *IEEE International Conference on Communications (ICC)*, 2017, pp. 1–7.

RÉFÉRENCES

The 10 Biggest Cloud Outages of 2017. Accessed : 2018-07-10.

Ayoubi, S., Assi, C., Narayanan, L. & Shaban, K. (2016a). Optimal polynomial time algorithm for restoring multicast cloud services. *IEEE Communications Letters*, 20(8), pp.1543–1546.

Ayoubi, S., Chen, Y. & Assi, C. (2016b). Towards promoting backup-sharing in survivable virtual network design. *IEEE/ACM Transactions on Networking*, 24(5), pp.3218–3231.

Bari, M., Boutaba, R., Esteves, R., Granville, L. Z., Podlesny, M., Rabbani, M. G., Zhang, Q. & Zhani, M. F. (2013). Data Center Network Virtualization : A Survey. *IEEE Communications Surveys Tutorials*, 15(2), pp.909-928.

Bo, L., Huang, T., SUN, X.-c., CHEN, J.-y. & LIU, Y.-j. (2014). Dynamic recovery for survivable virtual network embedding. *The Journal of China Universities of Posts and Telecommunications*, 21(3), pp.77–84.

Boutaba, R., Zhang, Q. & Zhani, M. F. (2013). Virtual Machine Migration : Benefits, Challenges and Approaches. Dans Mouftah, H. T. & Kantarci, B. (Éds.), *Communication Infrastructures for Cloud Computing : Design and Applications* (pp. 383-408). USA : IGI-Global.

CISCO. Concepts Cisco SDN. Accessed : 2019-01-10.

COHEN, G. Downtime, Outages and Failures - Understanding Their True Costs. Accessed : 2018-07-13.

ETSI. (2015a). Network Functions Virtualisation (NFV) ; Resiliency Requirements. Accessed : 2018-07-10.

ETSI. (2015b). ETSI GS NFV 003 V1.2.1 : Network Functions Virtualisation (NFV) ; Terminology for Main Concepts in NFV. Accessed : 2019-01-10.

Ghaleb, A. M., Khalifa, T., Ayoubi, S., Shaban, K. B. & Assi, C. (2016). Surviving multiple failures in multicast virtual networks with virtual machines migration. *IEEE Transactions on Network and Service Management*, 13(4), pp.899–912.

Guo, B., Qiao, C., Wang, J., Yu, H., Zuo, Y., Li, J., Chen, Z. & He, Y. (2014). Survivable virtual network design and embedding to survive a facility node failure. *IEEE Journal of Lightwave Technology*, 32(3), pp.483–493.

Harris, R. (2018). Fail-slow at scale : When the cloud stops working. Accessed : 2018-07-10.

Herrera, J. G. & Botero, J. F. (2016). Resource Allocation in NFV : A Comprehensive Survey. *IEEE Transactions on Network and Service Management (TNSM)*, 13(3), pp.518-532.

Hmaity, A., Savi, M., Musumeci, F., Tornatore, M. & Pattavina, A. (2016). Virtual network function placement for resilient service chain provisioning. *IEEE International Workshop on Resilient Networks Design and Modeling (RNDM)*, pp. 245–252.

IBM. IBM ILOG CPLEX Optimization Studio. Accessed : 2019-01-10.

informatique, S. (2010). Le livre b lanc du c loud Computing. Accessed : 2019-01-10.

Karra, K. & Sivalingam, K. M. (2018). Providing Resiliency for Service Function Chaining in NFV systems using a SDN-based approach. *2018 Twenty Fourth National Conference on Communications (NCC)*, pp. 1–6.

Lee, J., Ko, H., Suh, D., Jang, S. & Pack, S. (2017). Overload and failure management in service function chaining. *IEEE Conference on Network Softwarization (NetSoft)*, pp. 1–5.

Luizelli, M. C., Bays, L. R., Buriol, L. S., Barcellos, M. P. & Gaspary, L. P. (2015). Piecing together the NFV provisioning puzzle : Efficient placement and chaining of virtual network functions. *IFIP/IEEE International Symposium on Integrated Network Management (IM)*.

Mell, P. (2011). The NIST Definition of Cloud Computing. Accessed : 2019-01-10.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F. & Boutaba, R. (2016). Network function virtualization : State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1), pp.236–262.

ONF. Software-Defined Networking (SDN) Definition. Accessed : 2019-01-10.

Qu, L., Assi, C. & Shaban, K. (2016). Network function virtualization scheduling with transmission delay optimization. *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pp. 638–644.

Rabbani, M. G., Zhani, M. F. & Boutaba, R. (2014). On Achieving High Survivability in Virtualized Data Centers. *IEICE Transactions on Communications*, E97-B(1).

Racheg, W., Ghrada, N. & Zhani, M. F. (2017). Profit-driven resource provisioning in NFV-based environments. *IEEE International Conference on Communications (ICC)*, pp. 1–7.

Rahman, M. R. & Boutaba, R. (2013). SVNE : Survivable virtual network embedding algorithms for network virtualization. *IEEE Transactions on Network and Service Management*, 10(2), pp.105–118.

STRATO. STRATO. Accessed : 2019-01-10.

Technologies, H. UCAAS vs CPAAS - How to make sense of all those AAS terms. Accessed : 2019-01-10.

Xiao, A., Wang, Y., Meng, L., Qiu, X. & Li, W. (2014). Topology-aware virtual network embedding to survive multiple node failures. *IEEE Global Communications Conference (GLOBECOM)*, pp. 1823–1828.

Xu, J., Tang, J., Kwiat, K., Zhang, W. & Xue, G. (2012). Survivable virtual infrastructure mapping in virtualized data centers. *2012 IEEE Fifth International Conference on Cloud Computing*, pp. pp.196–203.

Yu, H., Anand, V., Qiao, C. & Sun, G. (2011). Cost efficient design of survivable virtual infrastructure to recover from facility node failures. *IEEE International Conference on Communications (ICC)*, pp. 1–6.

Zhang, Q., Zhani, M. F., Jabri, M. & Boutaba, R. (2014a). Venice : Reliable virtual data center embedding in clouds. *IEEE INFOCOM Proceedings*, pp. 289–297.

Zhang, Q., Zhani, M. F., Jabri, M. & Boutaba, R. (2014b). Venice : Reliable virtual data center embedding in clouds. *IEEE International Conference on Computer Communications (INFOCOM)*.

Zhani, M. F. & Boutaba, R. (2015). Survivability and Fault Tolerance in the Cloud. Dans *Cloud Services, Networking, and Management* (pp. pp.295–308). John Wiley & Sons, Inc. doi : 10.1002/9781119042655.ch12.