Protocole de routage pour l'architecture NDN

Elian Aubry

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Protocole de routage pour l'architecture NDN

THÈSE

présentée et soutenue publiquement le 19 Décembre 2017

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Elian Aubry

Composition du jury

Rapporteurs :	Pr. Mohamed Yacine GHAMRI-DOUDANE – *Professeur – Université de La Rochelle, France*
	Pr. Fabio MARTIGNON – *Professeur – Université de Bergame, Italie*
Examinateurs :	Dr. Olivier FOURMAUX – *Maître de conférences – UPMC Sorbonne Universités, France*
	Dr. Bertrand MATHIEU – *Directeur de recherche – Orange Labs, France*
	Pr. François CHAROY – *Professeur – Université de Lorraine, France*
Encadrants :	Pr. Isabelle CHRISMENT – *Professeur – Université de Lorraine, France*
	Dr. Thomas SILVERSTON – *Chercheur – NICT, Japon*

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

Je tiens tout d'abord à remercier mes encadrants, *Isabelle Chrisment* et *Thomas Silverston* pour leur aide ainsi que leur patience à mon égard durant ces quatre années.
Un grand merci à tous mes amis et collègues que j'ai eu la chance de côtoyer au sein du laboratoire *François*, *Eric*, *Anthéa*, *Gaëtan*, *César*, *Saïd*, *Valia*, *Daishi*, *Iñaki*,*Younes*, *Irfan*, *Saïf*, *Rémy*, *Kévin*, et bien d'autres.

Sans la possibilité de m'aérer l'esprit, cette thèse n'aurait pas abouti, je remercie donc tous mes amis et coéquipiers du club de baseball de Metz, ainsi que tous ceux qui m'ont permis de me changer les idées lorsque j'en éprouvais le besoin. En particulier *Emmy*, *Anaïs*, *Romain*, *Émeric*, *Tatiana*, *Morgane*, *Coco*, *Xavier*, *Quentin*, *Guy*, *Lothaire*, *Mary-Charlotte*, *Alice*, *Clémence*, *Hélène*, *Maxime*, *Sarah*, *Pauline*, *Baptiste*, *Margot*, *Pierre*, *Rems*, *Daniel*, *Bruno*, *Umberto*, *Kyle*, *Chad*, ainsi que tous ceux que j'aurais pu oublier mais qui me sont proches et chers. Énormément de reconnaissance envers ma chère et tendre *Emmy*, pour sa capacité à me supporter les mauvais jours lors de cette dernière année.

Enfin, une attention très particulière est portée à ma famille, sans laquelle je ne serais pas parvenu à terminer cette thèse. Merci pour le soutien de ma sœur, mon frère et mes parents, ainsi que pour leurs encouragements qui m'ont accompagné tout au long de cette grande, difficile et belle épreuve. Un grand mérite revient à mes parents, qui m'ont apporté, en plus de l'aide morale, l'aide financière pour parvenir à faire mes études dans de bonnes conditions. De plus, je tiens à encore une fois remercier ma mère pour son implication et les heures passées pour la lecture de mon manuscrit.

Je dédie cette thèse
à mes parents,
qui ont toujours cru en moi
et qui m'ont permis de réaliser mes projets.

Sommaire

Chapitre 1
Introduction **9**

1.1 Contexte et motivation de la thèse . . . 9
1.2 Problématique . . . 10
1.3 Contributions et plan de la thèse . . . 11

Chapitre 2
De l'Internet aux réseaux orientés contenu **13**

2.1 Introduction . . . 13
2.2 Architecture de l'Internet . . . 14
2.3 Communication de groupe – *multicast* réseau . . . 19
2.4 Réseaux de distribution de contenu . . . 20
2.5 Les réseaux pair à pair . . . 21
2.6 Les réseaux orientés contenu . . . 23
2.6.1 *Named-Data Networking (NDN) / Content-Centric Networking* (CCN) . . . 24
2.6.2 *Data Oriented Network Architecture* (DONA) . . . 27
2.6.3 *Publish-Subscribe Internet Technologies* (PURSUIT) . . . 28
2.6.4 *Network of Information* (NetInf) . . . 29
2.6.5 Autres architectures . . . 30
2.6.6 Comparaison des architectures orientées contenu . . . 31
2.7 Résumé . . . 35

Chapitre 3
Routage dans les réseaux orientés contenu **37**

3.1 Introduction . . . 37
3.2 Schémas de *forwarding* . . . 38
3.2.1 Forwarding réactifs . . . 38
3.2.2 Forwarding non-réactifs . . . 41

3.3 Schémas de routage . . . 44
3.3.1 Routage avec signalisation globale dans le réseau . . . 44
3.3.2 Routage avec signalisation locale dans le réseau . . . 49
3.3.3 Routage avec gestion centralisée de la signalisation . . . 51
3.4 Comparaison des différentes solutions . . . 53
3.5 Résumé . . . 54

Chapitre 4
SRSC : Protocole de routage basé sur SDN **57**

4.1 Introduction . . . 57
4.2 Intégration de SDN dans les ICN . . . 58
4.2.1 Intégration SDN-ICN avec IP . . . 58
4.2.2 Intégration SDN-ICN natif . . . 60
4.3 *SDN-based Routing Scheme for CCN/NDN* (SRSC) . . . 62
4.3.1 Principes de fonctionnement . . . 62
4.3.2 Phase d'amorçage (i.e. *Bootstrapping*) : . . . 67
4.3.3 Phase d'acheminement des règles (i.e. *Rules Forwarding*) : . . . 76
4.4 Résumé . . . 82

Chapitre 5
Expériences de simulation **83**

5.1 Introduction . . . 83
5.2 Environnement de simulation . . . 83
5.2.1 Paramètres . . . 84
5.2.2 Scénarios . . . 86
5.2.3 Métriques . . . 87
5.3 Résultats de l'évaluation . . . 88
5.4 Résumé . . . 94

Chapitre 6
Implantation dans NDNx : plateforme expérimentale

6.1 Introduction . . . 95
6.2 Plateforme expérimentale virtualisée . . . 96
6.3 Évaluation de SRSC via notre plateforme expérimentale virtualisée . . . 97
6.3.1 Démonstration de faisabilité de SRSC . . . 97
6.3.2 Évaluation de SRSC et comparaison avec NLSR . . . 100
6.4 Discussion . . . 108
6.5 Résumé . . . 109

Chapitre 7
Conclusion **111**

7.1 Résumé des contributions 112
7.2 Travaux futurs 112

Table des figures **115**

Liste des tableaux **119**

Bibliographie **121**

Chapitre 1

Introduction

Sommaire

1.1 Contexte et motivation de la thèse 9
1.2 Problématique 10
1.3 Contributions et plan de la thèse 11

1.1 Contexte et motivation de la thèse

L'Internet est un réseau de données mondial dont l'utilisation n'a cessé de croître depuis une trentaine d'années [1]. L'apparition de premiers services tels que la messagerie électronique (e-mail) ou le Web ont largement contribué au succès de ce réseau à la fin des années 1990 et à l'augmentation de son nombre d'utilisateurs. Plus récemment, de nouveaux services de distributions de contenu sont apparus ; au-delà des contenus textuels comme le Web ou e-mail, les utilisateurs pouvaient en effet accéder à des contenus multimédia type fichiers audio ou vidéo, d'abord en les téléchargeant, puis via des systèmes de diffusion de flux vidéos. Si, dans les années 2000, de nombreux utilisateurs téléchargeaient des fichiers audio (au format MP3) notamment via l'apparition de plateformes P2P tel que Napster [2], puis des fichiers vidéo via de nouveaux systèmes P2P type eDonkey [3] ou BitTorrent [4], de nouvelles plateformes de diffusion de vidéos ont ensuite vu le jour comme Youtube [5], Dailymotion [6] ou Netflix [7]. Ces dernières permirent aux utilisateurs de regarder des vidéos en *streaming*, c'est-à-dire en visionnant la vidéo au fur et à mesure de son téléchargement et non plus en téléchargeant tout le fichier au préalable.

A l'heure actuelle, les flux vidéo représentent la majeure partie du trafic total de l'Internet ; l'équipementier Cisco annonce que d'ici 2021, 82% du trafic sera dédié à la vidéo [8]. Les usages de l'Internet actuel ont considérablement évolué au fil des années, alors que l'architecture en elle-même a peu changé. En effet, si l'infrastructure s'est améliorée avec le déploiement de fibres optiques qui ont permis de démultiplier les débits, passant de quelques kb/s à plusieurs Gb/s [9], et avec l'apparition de réseaux sans-fil (WiFi) ou mobiles (3G, 4G, 5G) donnant accès à l'Internet depuis n'importe quel lieu [10], l'architecture de l'Internet n'a pas évolué et repose toujours sur la pile protocolaire TCP/IP.

Cette architecture de l'Internet s'appuie sur un réseau dit de moindre effort (*best effort*) où le modèle de communication de bout en bout permet aux hôtes d'accéder au service d'un autre hôte suivant le modèle client-serveur de l'Internet [11, 12, 13]. Ainsi avec TCP/IP, les datagrammes IP adressent des hôtes afin de les faire communiquer de bout en bout et le protocole TCP permet de fiabiliser les transferts [14].

Quand bien même des améliorations ont été proposées comme les communications de groupe (*multicast*) [15], les réseaux pairs-à-pairs (P2P) [16] ou les réseaux de distribution de contenu (CDN, *Content Distribution Networks*) [17], l'architecture TCP/IP n'a pas été conçue avec comme objectif de permettre la distribution de contenu à très large échelle et à un très grand nombre d'utilisateurs [18].

Les utilisateurs étant principalement intéressés par le contenu et non par sa localisation, de nouvelles architectures orientées contenu ont été proposées pour un Internet du futur [19, 20, 21, 22]. Ces architectures appelées génériquement *Information-Centric Networks* (ICN) se concentrent sur les données elles-mêmes et non sur les hôtes du réseau ; les données sont nommées, les hôtes ne le sont plus. L'acheminement des messages n'est donc plus réalisé en fonction de l'adresse de l'hôte comme cela est le cas avec l'architecture IP, mais en fonction du nom de la donnée. Ces nouvelles architectures sont conçues pour permettre la diffusion de contenu à large échelle, tout en ajoutant de nouvelles fonctionnalités dans le réseau comme les communications de type *anycast* [23], permettant de diriger les requêtes vers une ou plusieurs sources possédant le contenu. Elles permettent également aux entités du réseau (les routeurs) de stocker les données dans leurs mémoires afin de les retransmettre lors de futures requêtes. Ainsi, les contenus finissent par se rapprocher des utilisateurs, allégeant la charge du réseau, réduisant les délais pour les utilisateurs et améliorant leur qualité d'expérience.

En parallèle des travaux sur les réseaux orientés contenu, les réseaux logiciels, ou *Software-Defined Networks* (SDN), ont été définis pour apporter de la flexibilité dans la gestion des réseaux. Les réseaux logiciels permettent de découpler le plan de données (transmission des données) du plan de contrôle (routage) du réseau via l'utilisation d'une entité programmable appelée contrôleur. Les décisions de routage sont externalisées dans le contrôleur qui peut ainsi réaliser un routage fin, par flux ou par paquet. Cette flexibilité nous a semblé tout à fait adaptée pour fournir un nouveau plan de routage aux architectures orientées contenu.

1.2 Problématique

Dans cette thèse de doctorat, nous étudions les nouvelles architectures orientées contenu conçues pour être à la base d'un Internet du futur. Ces architectures orientées contenu adressent les données et non plus les hôtes du réseau, et cela implique que les mécanismes de routage utilisés dans les réseaux IP ne sont plus adéquats pour ces nouvelles architectures.

Ainsi, les architectures orientées contenu ne peuvent être totalement fonctionnelles sans plan de routage et ne peuvent donc être déployées pour remplacer l'architecture actuelle de l'Internet. Il est par conséquent essentiel de proposer de nouveaux mécanismes de routage qui soient adaptés

aux architectures orientées contenu. Ces nouveaux mécanismes doivent également tirer parti des nouvelles fonctionnalités mises à disposition par les architectures ICN, comme par exemple la mise en cache des contenus.

1.3 Contributions et plan de la thèse

Ainsi, dans cette thèse de doctorat, nous avons étudié les différents mécanismes de routage pour les architectures orientées contenu et nous proposons un nouveau protocole de routage pour l'architecture *Named-Data Networking* (NDN).

Les contributions de cette thèse sont multiples :

— Nous présentons un état de l'art des architectures orientées contenu, mais plus particulièrement des mécanismes de routage adaptés à ce nouveau type d'architecture.

— Nous présentons notre protocole de routage SRSC (*SDN-based Routing Scheme for NDN*).

— Nous évaluons les performances de SRSC à travers des expériences de simulation.

— Nous montrons par ces résultats qu'il n'est pas nécessaire d'équiper tous les nœuds d'une capacité de stockage. Ce résultat est important car il montre aux opérateurs qu'ils pourraient déployer cette architecture à des coûts d'infrastructure moindres (CAPEX).

— Nous avons déployé SRSC dans l'implantation de NDN appelée NDNx, et évalué ses performances sur notre plateforme expérimentale virtualisée sous Docker.

Dans ce manuscrit, nous commençons par introduire dans le Chapitre 2 l'Internet actuel et ses méthodes de distribution de contenu. Nous présentons et comparons ensuite les principales architectures orientées contenu proposées et justifions notre choix d'utiliser l'architecture NDN pour nos travaux. Un état de l'art des différentes solutions de routage présentes dans la littérature et adaptées à ces nouvelles architectures est réalisé dans le Chapitre 3.

Dans le Chapitre 4, Nous présentons notre protocole de routage SRSC, *SDN-based Routing Scheme for CCN/NDN*, qui est basé sur le paradigme SDN et permet le routage *anycast*, c'est à dire vers la copie du contenu la plus proche. Nous rappelons également le paradigme des réseaux logiciels, ainsi que les travaux connexes qui intègrent ces réseaux logiciels aux réseaux orientés contenu.

Nous évaluons les performances de notre protocole SRSC via des expériences de simulation avec le simulateur réseau NS-3 et son module ndnSIM dans le Chapitre 5. Ses performances sont mises en perspective avec l'inondation, technique initialement utilisée par NDN pour transmettre les requêtes.

Nous présentons notre implantation de SRSC dans NDNx ainsi que notre plateforme expérimentale virtualisée à travers Docker dans le Chapitre 6. Nous évaluons les performances de SRSC en le comparant à NLSR, devenu le protocole de routage disponible dans NDNx.

Enfin, nous concluons ce manuscrit dans le Chapitre 7, où nous résumons nos contributions et présentons des perspectives de travaux futurs. Nous listons également les différentes publications réalisées dans le cadre de ce doctorat.

Chapitre 2

De l'Internet aux réseaux orientés contenu

Sommaire

2.1 Introduction 13
2.2 Architecture de l'Internet 14
2.3 Communication de groupe – *multicast* réseau 19
2.4 Réseaux de distribution de contenu 20
2.5 Les réseaux pair à pair 21
2.6 Les réseaux orientés contenu 23
2.6.1 *Named-Data Networking (NDN) / Content-Centric Networking* (CCN) 24
2.6.2 *Data Oriented Network Architecture* (DONA) 27
2.6.3 *Publish-Subscribe Internet Technologies* (PURSUIT) 28
2.6.4 *Network of Information* (NetInf) 29
2.6.5 Autres architectures 30
2.6.6 Comparaison des architectures orientées contenu 31
2.7 Résumé 35

2.1 Introduction

L'Internet est un réseau de commutation de datagrammes qui offre un service de type *best-effort* (moindre effort), c'est-à-dire sans aucune garantie quant à l'acheminement des messages. Il est construit sur un modèle où les différents hôtes du réseau communiquent de bout en bout et est basé sur la pile protocolaire TCP/IP [24, 14]. Dans les années 2000, avec l'arrivée des technologies sans-fil tels que le WIFI, ou des télécommunications mobiles (3G/4G/5G), les hôtes peuvent désormais êtres mobiles et les utilisateurs connectés à l'Internet au travers des périphériques tels que des téléphones portables (smartphones) ou ordinateurs portables.

Pour permettre aux entités de communiquer entre elles, les hôtes de l'Internet utilisent le protocole IP et sont identifiés via leurs adresses IP. L'Internet repose ensuite sur le modèle client-serveur, où des clients-hôtes effectuent des requêtes vers des serveurs-hôtes. C'est le cas notamment du Web où des clients envoient leurs requêtes vers des serveurs web (ex : Facebook.com, Google.com) qui leur retournent les contenus (pages web dans ce cas).

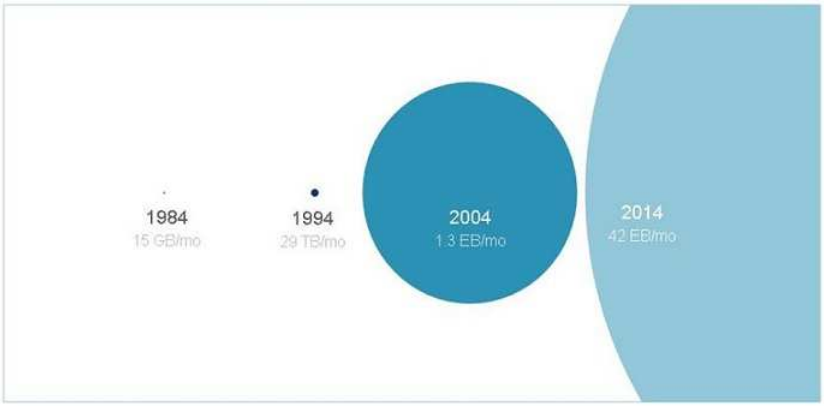


Figure 2.1 – Évolution de la quantité de trafic sur Internet entre 1984 et 2014 [1]

Ce modèle client-serveur prédominant au début de l'Internet a nécessité certaines adaptations avec la démocratisation de l'Internet. En effet, au delà des contenus textuels (web, email) de nouveaux contenus multimédias étaient consommés via téléchargement ou site de *streaming*, augmentant considérablement les volumes de trafic échangés. Comme le montre la Figure 2.1, le volume de trafic est passé d'environ 15 Go seulement en 1984 à des dizaines d'exa-octets par mois en 2014 (42 Eo en 2014 soit $42 * 10^{18}$ octets). Le trafic annuel de tout l'Internet d'il y a 30 ans correspond à peine au trafic mensuel d'un unique utilisateur en 2014.

L'architecture de l'Internet n'ayant pas été conçue à l'origine pour le transport de telles quantités de données, de nouvelles techniques ont été mises au point pour lui permettre de supporter cette charge de trafic. En effet, un serveur unique pourrait ne pas disposer d'assez de ressources si le contenu qu'il fournit se mettait à être extrêmement populaire et donc fréquemment demandé par de nombreux utilisateurs. Ces nouvelles techniques proposées permettent d'alléger la charge du serveur et de distribuer les contenus de manière efficace à l'échelle de l'Internet.

Dans ce chapitre, nous présentons plus en détail l'architecture de l'Internet, puis nous étudions les différentes solutions proposées pour permettre la distribution de contenu au sein de ce réseau. Notamment, nous présentons les techniques de communication de groupe, aussi appelées *multicast* réseau, puis nous évoquons les réseaux de distribution de contenu, ou *Content Delivery Networks* [17] et enfin les réseaux pair-à-pair [16]. Nous expliquons ensuite les concepts des réseaux orientés contenu, avant d'en présenter les principales architectures.

2.2 Architecture de l'Internet

L'architecture de l'Internet repose sur le modèle TCP/IP. Le protocole IP, utilisé pour permettre l'interconnexion de réseaux, a été conçu pour transmettre des messages (datagrammes) d'une source vers une destination. Il ne dispose en lui-même d'aucun mécanisme de fiabilité (réseau *best-effort*) mais peut profiter des services de bout en bout (contrôle de flux, contrôle d'erreurs) offert par la couche transport TCP. Les hôtes du réseau sont identifiés par une adresse unique, d'une longueur de 32 bits (IPv4) ou 128 bits (IPv6), appelée adresse IP. Cette dernière est hiérarchique et se compose de deux parties : l'adresse du réseau et l'adresse de la machine au sein de ce réseau.

Routage dans l'Internet

L'infrastructure de l'Internet comprend ainsi des routeurs qui sont des machines responsables d'acheminer les datagrammes vers leurs destinations. Pour cela, les équipements d'interconnexion utilisent des protocoles de routage [25] afin de calculer des tables de routage. Une fois ces tables de routage mises en place dans les routeurs, les datagrammes vont être transmis en fonction de leur adresse de destination à travers l'interface du routeur associée à cette adresse destination. Cette action consistant à transférer les datagrammes via les interfaces des routeurs est appelée *forwarding*. Ainsi, de proche en proche, routeur après routeur, le datagramme va se rapprocher de la destination jusqu'à l'atteindre. À l'échelle de l'Internet, on parle ainsi de routage saut à saut (*hop-by-hop routing*), puisque chaque routeur réceptionnant un datagramme IP va consulter à nouveau sa table de routage et le transmettre via une interface de sortie vers la destination.

Ces protocoles de routage peuvent être classés en deux catégories : à état de lien et à vecteur de distance. Par exemple, les protocoles dits à état de lien comme OSPF ou IS-IS [26] vont transmettre chaque information de routage à tous les routeurs du réseau, donnant l'ensemble des chemins possibles pour accéder à une destination. Tous les routeurs du réseau posséderont donc une vision commune de la topologie. Ensuite, parmi les chemins possibles, le plus court est calculé grâce à l'algorithme de Dijkstra [27]. Un fois le chemin le plus court obtenu, il sera choisi pour acheminer les datagrammes.

Les protocoles à vecteur de distance, comme RIP [28] ou IGRP [29], se contentent de transmettre uniquement le meilleur chemin pour atteindre une destination, calculé à l'aide de l'algorithme de Bellman-Ford [30]. Ainsi, chaque routeur diffuse à ses voisins uniquement le meilleur chemin pour atteindre une destination.

L'Internet est une interconnexion de plusieurs réseaux d'opérateurs qui sont administrés indépendamment les uns des autres. Les réseaux d'opérateurs peuvent eux-mêmes être découpés en Systèmes Autonomes (AS) qui auront chacun leur propre schéma d'administration. Ainsi, à l'intérieur de son propre réseau, ou de ses AS, un opérateur visera à optimiser les coûts d'acheminement des datagrammes et utilisera des protocoles de routage interne tels que OSPF, IS-IS ou RIP pour obtenir les plus courts chemins. Entre les différents AS, l'opérateur n'utilisera pas nécessairement les plus courts chemins, cela peut dépendre des partenariats économiques avec les autres opérateurs. Le protocole de routage inter-domaine utilisé est ainsi BGP (v4) [31], reposant sur les vecteurs de chemin et permettant aux opérateurs d'annoncer, si nécessaire, une partie seulement de leurs routes à leurs partenaires.

Par exemple, la Figure 2.2 représente trois AS exécutant les protocoles de routage internes (IS-IS, RIP et OSPF) et externe (BGP). Suite à cela, on peut imaginer qu'après accord entre les opérateurs ou choix de l'administrateur du réseau, BGP n'annonce pas les routes les plus courtes. Une conséquence possible serait que lorsque le Routeur3 de l'AS2 souhaite envoyer un datagramme à l'AS3, il ne puisse pas utiliser la route directe entre lui et le Routeur1, mais qu'il doive privilégier la route passant par le Routeur7 et donc l'AS1.

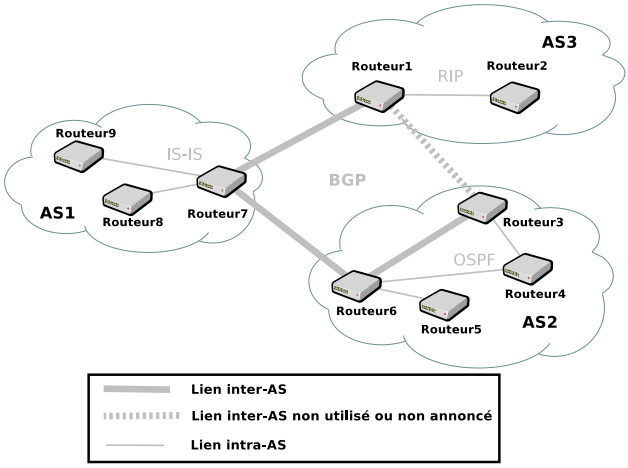


Figure 2.2 – Exemple de liaison entre plusieurs AS, exécutant les protocoles de routage internes (IS-IS, OSPF et RIP) et externe (BGP)

Système de nommage

Bien que les machines (hôtes ou routeurs) soient localisées par des adresses IP, les utilisateurs préfèrent, pour des raisons mnémotechniques, utiliser des noms pour les désigner. En effet, "www.wikipedia.fr" est plus facile à mémoriser que l'adresse IP "91.198.174.152", associée à ce serveur web. Lorsqu'un utilisateur effectue une requête web vers un nom de domaine, celui-ci sera converti en une adresse IP via le "*Domaine Name System*", ou système DNS [32]. Pour que la requête soit transmise vers une destination à travers le routage, il convient que cette requête soit exprimée sous une forme compréhensible par l'infrastructure réseau, c'est à dire une adresse IP.

Le système DNS est le mécanisme qui assure la traduction du nom de domaine en une adresse IP. Son système de nommage garantit l'unicité des noms. Il est hiérarchique, sous forme d'arbre de nommage, tout en étant indépendant de la hiérarchie de l'adresse IP. L'arbre de nommage est représenté en Figure 2.3. L'ICANN [33] est l'organisme responsable de l'attribution globale des adresses IP et de l'administration des noms de domaine de premier niveau.

En plus du nommage, le DNS offre un système de résolution. Chaque zone (ou nœud de l'arbre de nommage) gère son propre système DNS, permettant de gérer ses sous-zones et rediriger les requêtes vers les bonnes branches de l'arbre. Ainsi, si un resolveur local ne parvient pas à réaliser la conversion, il envoie, de manière itérative ou récursive, la requête aux différents serveurs faisant autorité dans l'arbre de nommage. Tout d'abord, au serveur de la racine, qui lui retourne l'adresse de serveur gérant la zone ".fr", puis au serveur DNS du ".fr" qui lui retourne l'adresse du serveur gérant la zone "wikipedia.fr", et enfin au serveur DNS de "wikipédia.fr" qui retourne l'adresse IP de son serveur web.

Le système DNS va ainsi agir tel un annuaire hiérarchisé et associer à un nom de domaine une adresse IP. Cette hiérarchie permet de passer à l'échelle et évite d'avoir à conserver l'ensemble des couples (nom de domaine ; adresse IP) sur tous les serveurs DNS, afin de distribuer la charge des requêtes dans les différents serveurs.

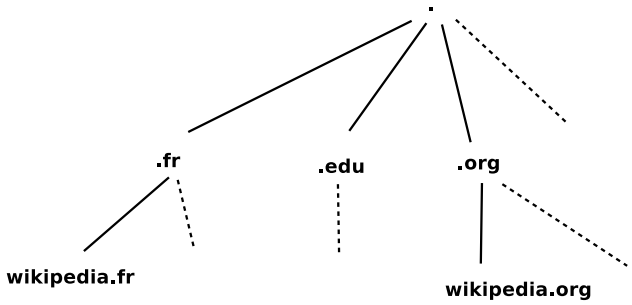


Figure 2.3 – Exemple de hiérarchie des serveurs de DNS

Accès aux contenus

Ainsi, l'apparition du web a permis l'accès à du contenu, même si ce dernier était dans un premier temps purement textuel. Chaque contenu du serveur web (page web, image, etc) est identifié via une URL ou *Uniform Ressource Locator* [34], laquelle URL est composée du nom de domaine (ex : www.wikipedia.fr) et du chemin d'accès sur le serveur (ex : `/wiki/loria/index.html`). Une requête web verra alors la partie domaine convertie via le service DNS en adresse IP et sera acheminée jusqu'au serveur web, lequel répondra en envoyant le contenu demandé (page web, image, etc.).

Ainsi, le web fut l'un des premiers moyens pour accéder à du contenu, d'abord textuel puis constitué de fichiers tels que des images ou d'autres formats comme des documents pdf, etc. Il était ensuite tout à fait possible de se servir du web pour transmettre des fichiers audio/vidéo, en téléchargement tout d'abord et plus récemment, en lecture en directe (*streaming*). Le web a ainsi fortement contribué à la démocratisation de l'Internet et à son utilisation quotidienne pour un très grand nombre d'utilisateurs dans le monde. C'est notamment l'accès à des nouveaux contenus et informations qui a permis ce développement.

La Figure 2.4 schématise le fonctionnement de l'Internet. Dans celui-ci, un client souhaite accéder à l'article de l'encyclopédie en ligne Wikipédia, concernant le laboratoire du Loria (message 1). Le resolveur du client est interrogé pour obtenir l'adresse IP associée au serveur de Wikipédia (message 2). Ici, elle est présente dans son cache. Dans le cas contraire, le système DNS est sollicité, en remontant dans la hiérarchie des DNS et ce, jusqu'à retrouver cette adresse IP dans un cache ou dans le DNS du domaine visé, comme cela est schématisé en Figure 2.5. Une fois l'adresse IP obtenue (message 3), le datagramme IP qui contient la requête web est envoyé vers le serveur (message 4). Les routeurs qui ont calculé leurs tables de routage acheminent ce datagramme vers la destination. Lorsque le serveur web de "wikipedia.fr" reçoit la requête du client (message 5), il retourne la réponse qui contient la page web demandée (message 6 et 7). Il est utile de préciser que la réponse ne prendra pas nécessairement le même chemin de retour que celui suivi par la requête.

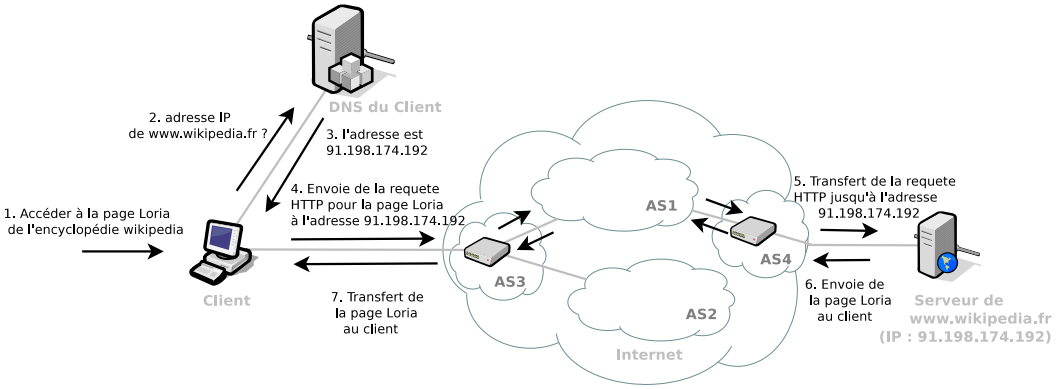


Figure 2.4 – Principe de fonctionnement du protocole IP à travers un exemple

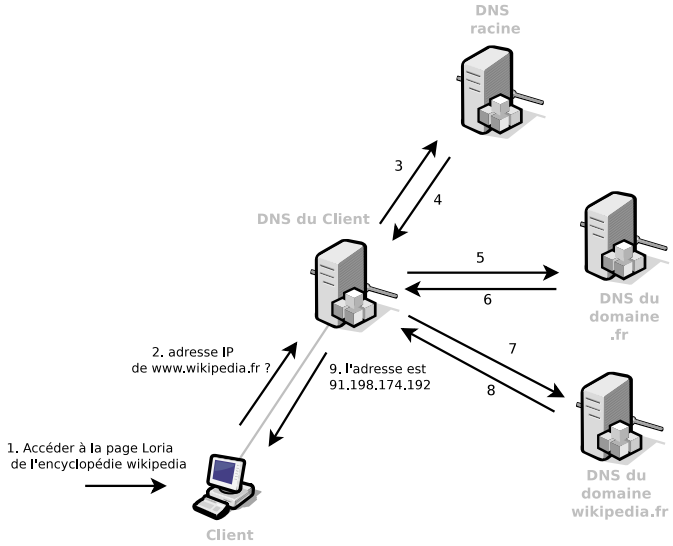


Figure 2.5 – Principe de fonctionnement des serveurs de DNS [35]

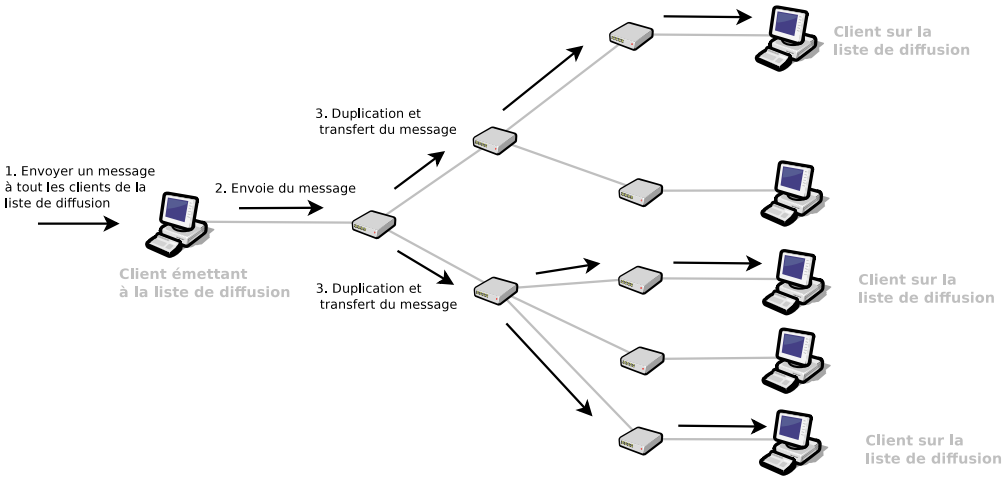


Figure 2.6 – Principe de fonctionnement du multicast

2.3 Communication de groupe – *multicast* réseau

Alors que la demande en contenu se faisait de plus en plus forte sur l'Internet, il a été nécessaire de proposer des solutions pour permettre une distribution de contenu à des groupes d'utilisateurs. L'Internet étant un réseau où les communications s'effectuent point-à-point entre un client et un serveur, si un contenu devient très populaire, il est donc demandé de nombreuses fois, obligeant le serveur à envoyer plusieurs fois les mêmes données, consommant des ressources système (CPU, mémoire) et réseau (bande passante, file d'attente des routeurs).

Les communications de groupe, ou multi-points (*multicast*) [15], permettent à l'émetteur de n'envoyer qu'une seule fois les données qui seront, si nécessaire, dupliquées suivant un arbre de distribution par les équipements du réseau (les routeurs), dans le but d'être distribuées à plusieurs utilisateurs. Ainsi, chaque donnée n'est transmise qu'une seule fois et ce sont les routeurs qui ont la charge de dupliquer les contenus vers les utilisateurs finaux. Les ressources du serveur sont alors préservées puisqu'une seule copie du contenu est transmise par celui-ci. Chaque lien du réseau est également parcouru par une seule copie, économisant également les ressources du réseau.

Afin de gérer la création de groupes d'utilisateurs, le protocole IGMP (*Internet Group Management Protocol*) [36] est utilisé. Il permet aux hôtes du réseau d'annoncer à leur routeur leurs abonnements à des groupes de diffusion *multicast*. Ensuite, des protocoles de routage comme PIM (*Protocol Independent Multicast*) [37] génèrent les arbres de diffusion, permettant d'acheminer les données aux groupes d'utilisateurs.

Ce type de communications multi-points trouve notamment tout son sens pour des applications de groupe comme la diffusion de télévision (c'est à dire du *streaming* vidéo). La Figure 2.6 représente ce mode de communication à travers l'envoi d'un message d'un utilisateur vers trois récepteurs appartenant à un même groupe *multicast*.

Cependant, si le *multicast* peut être déployé au sein du réseau d'un opérateur, comme pour la diffusion de la télévision, il apparaît alors complexe de transmettre un flux à un autre opérateur qui pourrait avoir l'effet de se démultiplier dans le réseau du concurrent. Ainsi, le *multicast* inter-domaine n'a pu être déployé à l'échelle de l'Internet, notamment pour des raisons économiques et politiques entre opérateurs [38, 39].

Une autre raison invoquée pour le non déploiement du *multicast* réseau, est qu'il nécessite de maintenir des états dans les routeurs pour dupliquer les messages, c'est à dire que la complexité est déplacée dans le cœur du réseau, et non plus en bordure du réseau comme cela est généralement le cas. Ainsi, cette approche va à l'encontre d'un des principes de l'Internet qui est de conserver un cœur de réseau relativement simple et la complexité au niveau des hôtes. Ce changement de paradigme a freiné son déploiement et, associé aux raisons économiques et politiques précédemment évoquées, empêché son déploiement à l'échelle de l'Internet.

2.4 Réseaux de distribution de contenu

Pour diminuer la charge des serveurs et réduire la distance entre les utilisateurs et les contenus, les réseaux de distribution de contenu, ou *Content Delivery Networks* (CDN) [17], se sont développés. Ces CDN ont pour but de répliquer les contenus disponibles sur le serveur d'origine à plusieurs endroits dans l'Internet, sur des serveurs qui leur sont propres.

Grâce à un mécanisme de routage, les messages, au départ adressés au serveur d'origine, vont être envoyés vers le serveur du CDN le plus proche. Ce principe de routage, appelé *anycast* permet donc à une même requête envoyée depuis des endroits géographiquement différents, d'être transmise à deux serveurs différents.

Deux méthodes sont principalement mises en place [40], l'utilisation d'une adresse IP *anycast* ou la redirection grâce au DNS. Avec l'adresse IP *anycast*, comme cela est le cas dans le CDN CloudFlare [41], une même adresse IP est attribuée au serveur d'origine ainsi qu'aux différents serveurs du CDN ; le chemin à suivre est alors celui donné par le protocole de routage. La méthode par résolution DNS, utilisée par le pionnier Akamai [42], consiste à avoir des couples (adresse IP ; nom de domaine) différents dans les DNS, permettant d'envoyer vers le serveur original ou un de ceux du CDN en fonction de la localisation. Dans cette méthode, chaque serveur a une adresse IP différente.

La Figure 2.7 représente le mode de fonctionnement d'un CDN utilisant la méthode de redirection DNS. Un client souhaite accéder à la page de l'encyclopédie Wikipédia concernant le Loria (message 1). Le client interroge donc son DNS (message 2). Lors de sa requête, le DNS du serveur (ou d'un niveau supérieur) récupère l'adresse IP du serveur le plus pertinent en fonction de l'origine de la requête. Dans cet exemple, l'adresse du CDN est renvoyée (message 3), permettant d'acheminer la requête vers ce dupliqua (message 4 et 5). Le contenu est alors retourné à l'utilisateur (messages 6 et 7) de manière totalement transparente.

En répliquant les contenus à plusieurs endroits, les serveurs du CDN vont permettre d'alléger la charge du serveur d'origine et en rapprochant le contenu des utilisateurs, de diminuer les délais et améliorer la qualité d'expérience des utilisateurs.

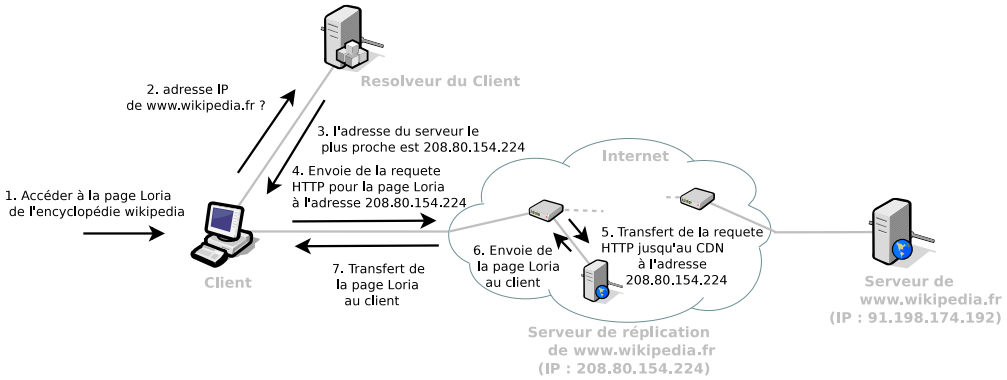


Figure 2.7 – Principe de fonctionnement des réseaux de distribution de contenus

2.5 Les réseaux pair à pair

Les réseaux pair-à-pair, ou *peer-to-peer* (P2P) [16] ont également été proposés pour permettre d'améliorer le partage de contenus volumineux dans l'Internet. Dans les réseaux P2P, chaque entité du réseau, appelée pair, est à la fois client et/ou serveur et peut tout aussi bien transmettre des données vers d'autres utilisateurs (rôle serveur) qu'en télécharger (rôle client). Ces architectures pair-à-pair peuvent elles-mêmes être centralisées comme Napster [2], qui repose sur l'utilisation d'un serveur central pour recenser tous les pairs et leurs fichiers, afin de mettre en relation un pair effectuant une requête vers un pair possédant le contenu recherché. D'autres architectures sont quant à elles décentralisées comme par exemple Gnutella [43], qui inonde le réseau de requêtes pour détecter les contenus disponibles. Une autre approche décentralisée repose quant à elle sur l'utilisation de tables de hachage distribuées (DHT), ou chaque pair rejoint une topologie d'overlay virtuelle et est responsable d'un espace d'adressage pour associer un contenu clé avec un pair le possédant. C'est par exemple le cas de Kademlia [44].

Les réseaux P2P sont particulièrement utilisés dans la distribution de fichier à un grand nombre de pairs, comme par exemple le système BitTorrent [4]. Les contenus populaires seront ainsi retransmis par un grand nombre de pairs qui ont eux-mêmes téléchargé le contenu, allégeant la charge d'un serveur unique et permettant la distribution de la charge totale sur tout le réseau. BitTorrent en est l'un des meilleurs exemples. La distribution de contenu étant elle même effectuée lorsqu'un pair rejoint un torrent, c'est à dire une topologie virtuelle dans laquelle tous les pairs possèdent des morceaux de fichiers, appelés *chunk*, qui sont échangés entre pairs du réseau. Sur la Figure 2.8, un pair souhaite accéder à un fichier X (message 1). Il émet sa requête dans le réseau (message 2). Il va pouvoir récupérer chez chacun des pairs, présents dans le réseau P2P et possédant le fichier, un *chunk* du fichier X, c'est à dire effectuer plusieurs téléchargements de morceaux du fichier en parallèle. Ceci va avoir comme effet de réduire la vitesse de téléchargement puisqu'un seul pair ne serait plus l'unique goulot d'étranglement du téléchargement (messages 3 et 4).

Dans le système BitTorrent, représenté en Figure 2.9, le client récupère auprès d'un *tracker* la liste des pairs possédant le contenu désiré. Ce *tracker* peut être partagé entre les différents pairs du réseau ou centralisé dans un serveur (serveur web). Une fois la liste récupérée, le pair rejoint le torrent et peut échanger directement des *chunks* de fichiers avec les autres pairs du réseau. Afin d'amorcer les échanges, BitTorrent repose également sur l'utilisation de deux algorithmes principaux : (i) le *choke algorithm* pour attribuer des morceaux de fichiers aléatoirement à des pairs, ce qui va permettre aux nouveaux pairs de commencer leur collection de *chunks* et de les échanger ; (ii) l'algorithme d'ordonnancement de téléchargement des *chunks rarest-first*, c'est à dire que les pairs choisissent de télécharger en premier le bloc le plus rare dans le torrent afin de préserver l'intégrité du fichier dans la durée.

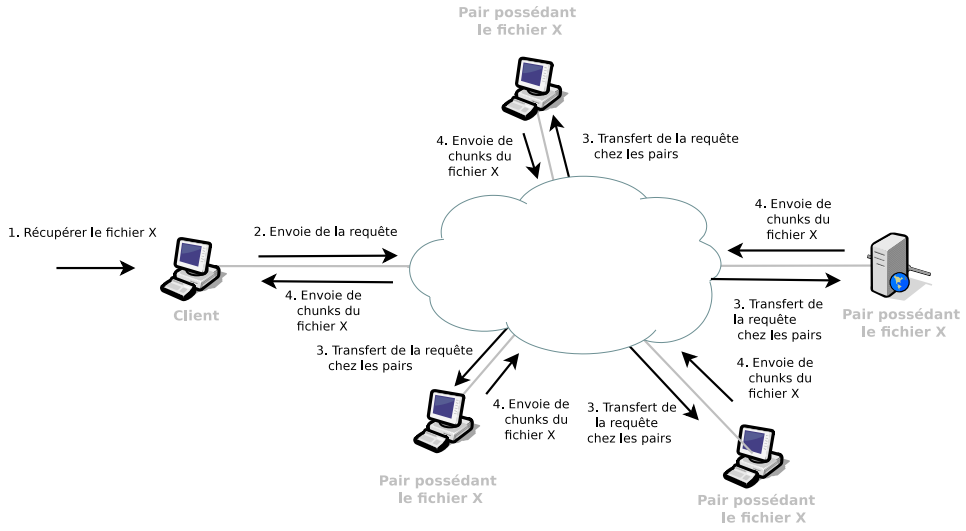


Figure 2.8 – Principe de fonctionnement du réseau Pair-à-Pair pour l'échange de fichier

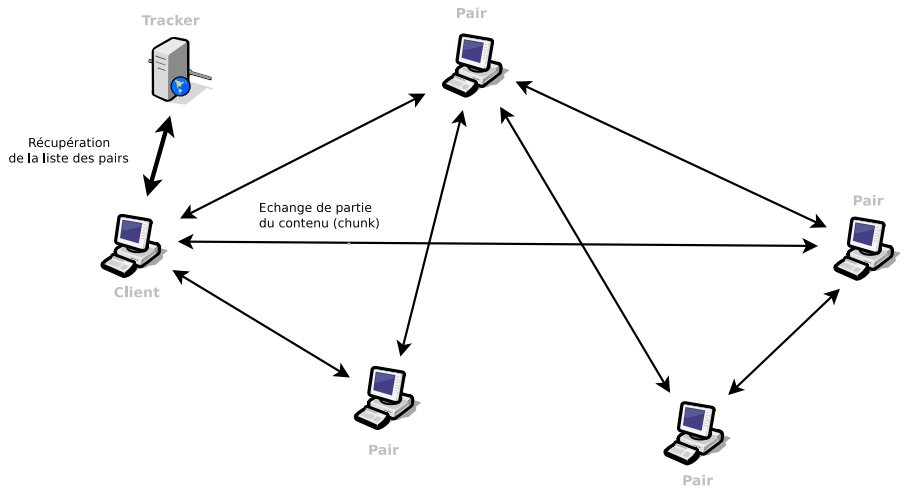


Figure 2.9 – Principe de fonctionnement du réseau Pair-à-Pair avec le système BitTorrent [4]

2.6 Les réseaux orientés contenu

Comme nous le rappelions dans l'introduction de ce chapitre, le volume de trafic de l'Internet est en augmentation perpétuelle et est constitué principalement par du contenu. En effet, le contenu vidéo (fichier, *streaming*, etc.) représente 73% du trafic consommateur de l'Internet en 2016 et jusque 82% en 2021 [8]. Ainsi, l'un des grands challenges de l'Internet est de permettre le transport massif de contenu, ce que les propositions présentées précédemment se sont efforcées d'apporter.

Toutefois, une nouvelle approche consiste à ne pas continuer à bâtir sur l'Internet existant et proposer des solutions qui ne peuvent être totalement déployées (*multicast* réseau), ou ne sont pas optimales en matière de consommation des ressources réseaux (CDN, P2P). Cette nouvelle approche consiste plutôt à refonder les bases d'une architecture pour un Internet du futur. Ainsi, de nouvelles architectures orientées contenu ont été proposées afin de disposer d'architectures réseaux adaptées aux nouveaux usages actuels, à l'accès massif aux différentes formes de contenu ainsi qu'à leur distribution. On parle de façon générique de réseaux ou architectures orientées contenu, ou bien *Information Centric Networks* (ICN) [45].

Dans cette nouvelle approche, le contenu est au cœur des communications et non plus les hôtes comme c'est le cas dans l'architecture actuelle de l'Internet avec les protocoles TCP/IP. Les architectures orientées contenu reposent sur des modèles de communication hôte-à-donnée, ou *content centric*, et non plus sur le modèle de communication de bout en bout de l'Internet, ou *host centric*. Ce sont bien les données qui seront au cœur de l'architecture et de ses mécanismes.

Ces nouvelles architectures reposent également sur un réseau de caches, où toutes les entités du réseau (hôtes ou nœuds) possèdent une capacité de stockage pour conserver les données. Ce principe rappelle quelque peu les réseaux P2P où les contenus finiront par se propager plus proches des utilisateurs ou être répliqués en plusieurs points du réseau, évitant le traditionnel goulot d'étranglement d'un serveur unique. De plus, les architectures ICN adressent maintenant les contenus et non les hôtes. Ainsi, n'importe quelle entité du réseau pourra répondre à une requête si elle dispose du contenu. Tous ces changements (réseau de caches et *content-centric*) constituent un changement de paradigme important par rapport à l'Internet actuel. Plusieurs architectures orientées contenu ont alors été proposées et seront présentées dans la suite de ce chapitre.

Ces architectures étant totalement différentes de l'architecture actuelle de l'Internet, elles proposent de nouvelles techniques originales de nommage, routage, contrôle des ressources ou dissémination des données et ont toutes leurs spécificités. Nous présenterons dans un premier temps les différentes architectures orientées contenu, en commençant par la plus emblématique d'entre elle, l'architecture *Named-Data Networking*, que nous utiliserons par la suite dans nos travaux. Nous comparerons ensuite ces architectures en mettant en évidence leurs points communs et différences.

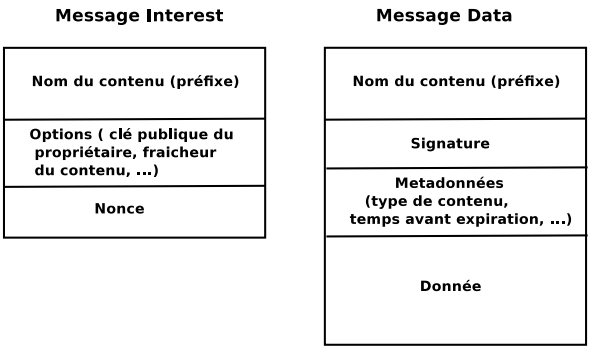


Figure 2.10 – Corps des messages *Interest* et *Data* utilisés dans NDN

2.6.1 *Named-Data Networking (NDN) / Content-Centric Networking* (CCN)

L'architecture NDN [20] a été initialement proposée via le projet NDN [46] et par Van Jacobson en 2009. Étant à la base intitulé CCN, ce projet a par la suite été décliné en deux autres projets (*fork*) : (i) le projet CCN, géré par le *Palo Alto Research Center* (PARC) [47] et dont l'architecture est propriétaire ; (ii) le projet NDN de la NSF, géré par UCLA et dont le code source est libre (*Open source*). Si les deux projets partagent la même architecture, ils ont ensuite évolué séparément au gré des différents contributeurs et objectifs des différents instituts [48]. Nous utiliserons par la suite principalement la dénomination NDN. Toutefois, nous pourrons utiliser le terme CCN de façon équivalente.

Comme pour tout réseau orienté contenu, pour NDN/CCN les données sont à la base des échanges. Chaque contenu est identifié par un nom -ou préfixe- qui a une structure hiérarchique, à la manière des URIs [49]. Ce nommage présente l'avantage d'avoir une signification sémantique pour les utilisateurs. Les préfixes NDN se présentent alors telles des URIs sous la forme suivante : `/loria/exemple/video/chat.avi`. Contrairement au système de nommage plat, le système de nommage hiérarchique offre la possibilité de pouvoir être agrégé. Par exemple `/loria/exemple/video/chat.avi/00001` et `/loria/exemple/video/chat.avi/0002`. identifient deux données qui peuvent être agrégées par le même préfixe. Il est également possible de passer à un nommage plat en fixant un préfixe et appliquant une fonction de hachage à la donnée.

Dans l'architecture NDN/CCN, lorsqu'un utilisateur souhaite obtenir un contenu, il envoie un message *Interest* dans le réseau, lequel lui sera répondu par un message *Data* possédant les données. Ces deux messages sont représentés sur la Figure 2.10. Les *Interest* contiennent le nom de la donnée, c'est à dire le préfixe de la donnée, ainsi qu'une *nonce* et plusieurs champs d'options possibles comme la durée de validité de la requête ou la clé publique du propriétaire qui a signé le message. Les messages *Data* sont quant à eux composés du nom de la donnée (le même préfixe que le message *Interest*), du contenu en lui même et de la signature du contenu. Ces messages laissent au réseau la tâche importante de résoudre les noms des requêtes et de permettre l'acheminement des données vers l'utilisateur, c'est-à-dire de router les messages dans le réseau. L'architecture NDN étant elle même un réseau de caches, plusieurs copies d'un même contenu peuvent être présentes dans différents nœuds du réseau. Les données peuvent ainsi provenir de différents points du réseau et aucune donnée n'est associée à priori à une localisation dans le réseau.

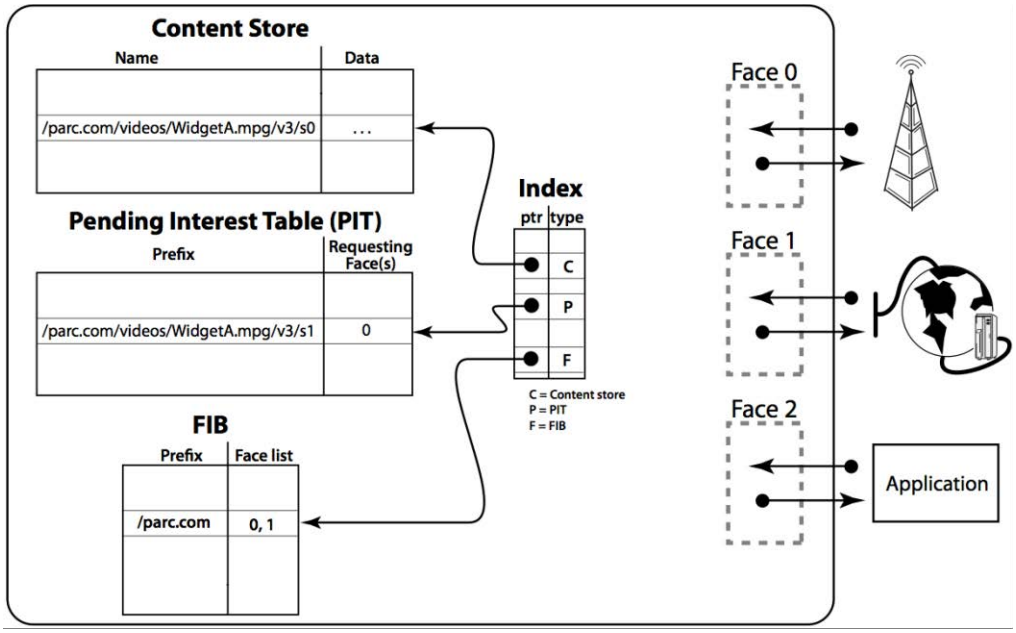


Figure 2.11 – Schéma des composants du nœud NDN [50]

Chaque nœud NDN comprend trois composants principaux représentés Figure 2.11 : (i) une table de transmission ou *Forwarding Information Base* (FIB) ; (ii) une table d'intérêt ou *Pending Interest Table* (PIT) ; (iii) et une mémoire cache appelée aussi *Content Store* (CS).

La FIB conserve les entrées où doivent être transmis les messages *Interest* vers des sources de contenus potentielles. La PIT permet à un nœud de conserver la liste des messages *Interest* qu'il a transmis et les interfaces sur lesquelles ils ont été reçus, de manière à éviter les retransmissions inutiles. Les entrées de la PIT sont aussi utilisées pour que les messages *Data* puissent emprunter le chemin inverse des messages *Interest*. Finalement, le CS est une mémoire cache qui permet de stocker des copies locales des contenus. Chaque nœud NDN pourra ainsi répondre directement aux requêtes (*Interest*) s'il possède les contenus dans son *Content Store*.

Les nœuds du réseau pouvant conserver une copie locale des contenus qui ont transité à travers eux, les contenus vont ainsi être disséminés dans le réseau et être plus proches des utilisateurs. Ainsi, les *Interest* vont potentiellement rencontrer un nœud plus proche possédant le contenu et ne pas être propagés jusqu'au serveur d'origine, ce qui va réduire la charge sur ce serveur et limiter l'impact d'une recherche dans le réseau. Les délais d'accès au contenu seront également réduits pour les utilisateurs et leur qualité d'expérience (délai, débit) en sera améliorée. La gestion de la mémoire cache des nœuds a été largement étudiée et de nombreux mécanismes ont été proposés pour décider quels contenus stocker en mémoire, et lesquels supprimer lorsque la mémoire est pleine [51, 52, 53, 54, 55].

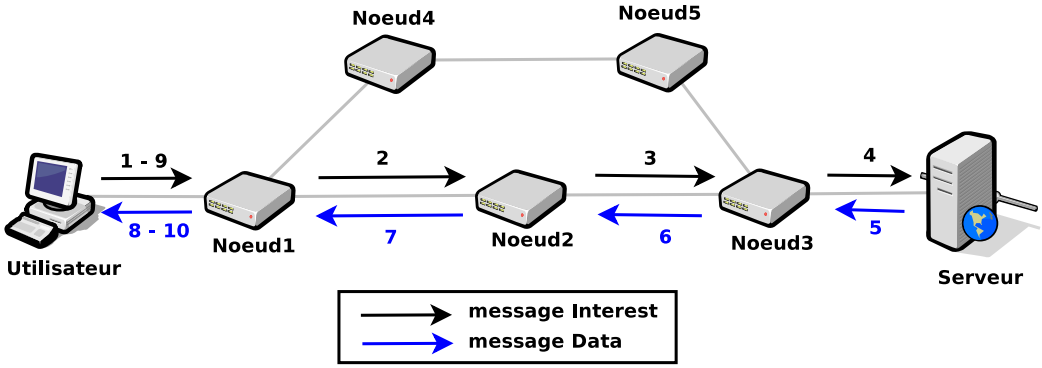


Figure 2.12 – Architecture CCN/NDN

Les grands principes du fonctionnement de l'architecture NDN/CCN sont représentés sur la Figure 2.12. Lorsqu'un utilisateur veut accéder à un contenu, il publie/envoie un message *Interest* (message 1). Quand le Noeud1 reçoit cet *Interest*, plusieurs cas de figure sont possibles : (i) soit il connaît le prochain nœud pour transmettre le message car une entrée est déjà dans sa FIB, auquel cas il lui transmet donc la requête (message 2) ; (ii) Soit il ne possède pas la règle pour atteindre ce contenu et va transmettre cette requête sur toutes ses autres interfaces (*flooding*). Dans notre exemple, on part du principe que les règles sont déjà dans les tables de transmission (FIB) des nœuds.

Le Noeud2 l'envoie à son tour au Noeud3 (message 3) qui la transfère au serveur (message 4). Ce dernier répond grâce à un message *Data* transportant la donnée. Ce message *Data* va utiliser le chemin inverse de l'*Interest* pour arriver à l'utilisateur (messages 5 à 8), car chaque nœud a conservé cette entrée dans sa PIT.

Tous les nœuds ayant retransmis le message *Data* vont pouvoir conserver une copie locale dans leurs mémoires cache (CS) et ce, en fonction des mécanismes de cache qui ont été instanciés sur chacun des nœuds.

Ainsi, lorsqu'un utilisateur émet à nouveau une requête pour ce même contenu (message 9), le Noeud1 va pouvoir utiliser sa copie locale pour répondre (message 10). C'est ce principe qui permet au contenu d'être propagé au plus près des utilisateurs, de réduire le délai d'accès au contenu et de limiter la portée d'une requête afin de ne pas surcharger le réseau.

Dans cette thèse, nous nous intéressons au routage des contenus dans cette architecture, routage qui doit être basé sur les noms puisque ceux-ci identifient les contenus demandés. En effet, lorsque nous avons débuté ces travaux, l'architecture NDN ne disposait d'aucun mécanisme de routage pour remplir les FIB des nœuds du réseau. Ainsi, lorsqu'un message *Interest* était transmis, les nœuds le retransmettaient sur toutes leurs interfaces, inondant le réseau de requêtes et consommant les ressources du réseau. Cependant, l'inondation ne peut être envisagée à l'échelle de l'Internet avec un nombre important de requêtes. Nous avons ainsi proposé un protocole de routage adapté à l'architecture de NDN. Notre protocole SRSC, présenté au chapitre 4, permet de transmettre une requête vers la copie la plus proche sans inonder le réseau.

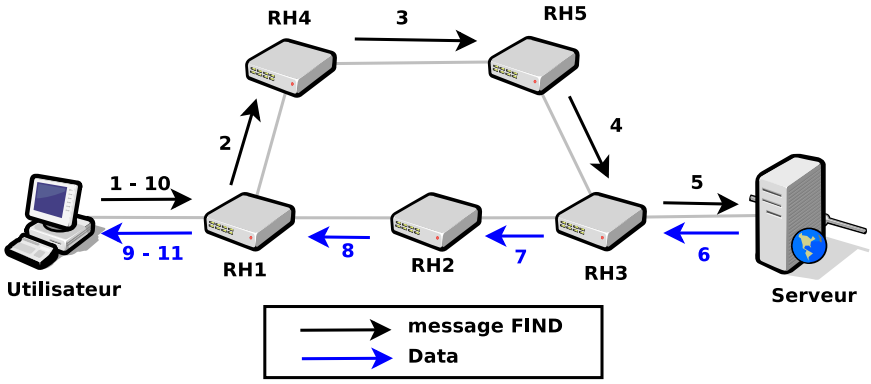


Figure 2.13 – Architecture DONA

2.6.2 *Data Oriented Network Architecture* (DONA)

DONA [19] est à notre connaissance la toute première architecture orientée contenu. Elle a été conçue par l'Université de Californie - Berkeley, et s'inscrit dans la continuité de l'architecture TRIAD (Translating Relaying Internet Architecture integrating Active Directories) [56]. Elle repose principalement sur l'utilisation de nouveaux mécanismes de nommage des contenus puisqu'elle utilise toujours l'architecture TCP/IP.

Afin d'identifier les contenus, DONA remplace le système de résolution de noms (DNS) traditionnel par un système de résolution de noms qui suit un plan de nommage plat. Chaque contenu est alors désigné par la concaténation de deux identifiants : (i) le premier identifiant correspond au résultat d'une fonction de hachage de la clé publique de l'utilisateur ayant proposé le contenu (le publieur) ; (ii) le second identifiant est un nom choisi par l'utilisateur. Les nœuds, appelés *Resolution Handlers* (RH), sont hiérarchisés et à chaque enregistrement d'un nouveau contenu, le RH prévient ses pairs ainsi que ses parents de sorte à propager l'information et permettre la résolution de noms et le routage par la suite.

Chaque nœud de l'architecture DONA possède ainsi une table de transmission lui permettant d'envoyer les messages au prochain nœud, offrant de ce fait la possibilité de transmettre les requêtes vers la source la plus proche. Mais ne disposant pas de système d'annonces pour gérer les données stockées en cache, DONA se contente d'envoyer les requêtes vers le serveur le plus proche. Une fois la donnée localisée, elle est acheminée jusqu'à la destination via des communications point-à-point entre récepteur et source via l'Internet actuel.

La Figure 2.13 illustre le principe de fonctionnement de l'architecture DONA. Un utilisateur émet un message FIND pour accéder à une donnée stockée dans le serveur (message 1). Le nœud RH1 ne connaît pas de chemin vers cette donnée, il transmet donc la requête FIND à son parent RH4 (message 2), qui la transmet à son pair RH5 (message 3), car il n'a ni règles pour accéder à la donnée, ni parent. RH5 connaît la route pour accéder à la donnée grâce aux annonces générées auparavant par le serveur, la requête est donc transmise vers le serveur (message 4 et 5). Le serveur répond à l'utilisateur en envoyant la donnée via une communication client/serveur classique de l'Internet, la donnée n'empruntant pas forcément le même chemin que la requête (message 6 à

9). Les nœuds intermédiaires (RH1, RH2 et RH3) peuvent décider de stocker dans leur mémoire cache les contenus qu'ils acheminent, lorsque l'utilisateur émettra une prochaine requête pour un de ces contenus (message 10), RH1 répondra avec la copie qu'il possède (message 11).

L'architecture DONA repose encore sur l'architecture actuelle de l'Internet, mais lui apporte une nouvelle résolution de noms qui vont permettre la distribution effective de contenu en allégeant la charge du réseau (plusieurs serveurs, mise en cache, etc.).

2.6.3 *Publish-Subscribe Internet Technologies* (PURSUIT)

L'architecture PURSUIT [21], qui fait suite à son prédécesseur PSIRP (Publish-Subscribe Interest Routing Paradigm) [57], a été proposée par le projet de recherche européen du même nom (FP7 EU PURSUIT [58]). Cette architecture utilise un modèle publication-souscription (*publish-suscribe*) pour délivrer les contenus.

PURSUIT est réalisée grâce à trois composants : le *rendez-vous*, le *topology manager*, et le *forwarding*. Le *rendez-vous* fait le lien entre les souscriptions et les publications. Le *topology manager* gère la topologie et crée les routes entre les publieurs et les souscripteurs de sorte à acheminer les contenus. Les *topology managers* exécutent un algorithme de routage distribué (type OSPF [59]) dans le but de découvrir les différentes routes possibles. Enfin le *forwarding* permet d'acheminer les données en suivant la route mise en place par son *topology manager* jusqu'à atteindre le souscripteur.

Pour nommer les informations, deux identifiants sont utilisés : le *scope ID* pour définir la catégorie de l'information, comme par exemple une musique, une image ou une vidéo, et le *rendez-vous ID* qui identifie l'objet en lui-même grâce à un nommage plat. Pour être plus visible et améliorer sa portée, un contenu peut être publié plusieurs fois avec des *scope ID* différents. Ceci veut dire qu'un contenu a un *rendez-vous ID* mais peut posséder plusieurs *scopes ID*.

Ce mode de nommage plat nécessite donc une entité permettant la résolution des noms. Ceci est effectué par les *rendez-vous*, qui vont décider de la route à suivre pour chaque requête.

Comme illustré par la Figure 2.14, le publieur annonce un nouveau contenu à son nœud de *rendez-vous* le plus proche, localisé grâce à une table de hachage distribuée ou *Distributed Hash Table* (DHT) présente dans les entités de *forwarding* (messages 1 et 2). Lorsqu'un souscripteur est intéressé par un contenu, il contacte son nœud de *rendez-vous* local en précisant le contenu souhaité (messages 3 et 4). Ce dernier va remonter dans la chaîne de *rendez-vous* afin de trouver le contenu (messages 5, 6 et 7). Une fois trouvé, le *rendez-vous* contacte le *topology manager* pour le configurer et créer la route entre le souscripteur et le publieur possédant le contenu (messages 8 et 9). Pour cela, le *topology manager* utilise un protocole de routage distribué (type OSPF) afin de calculer le chemin. Une fois que le publieur est averti de la nouvelle route mise en place (messages 10 et 11), il utilise sa fonction de *forwarding* pour transférer le contenu jusqu'au souscripteur (messages 12, 13, 14 et 15).

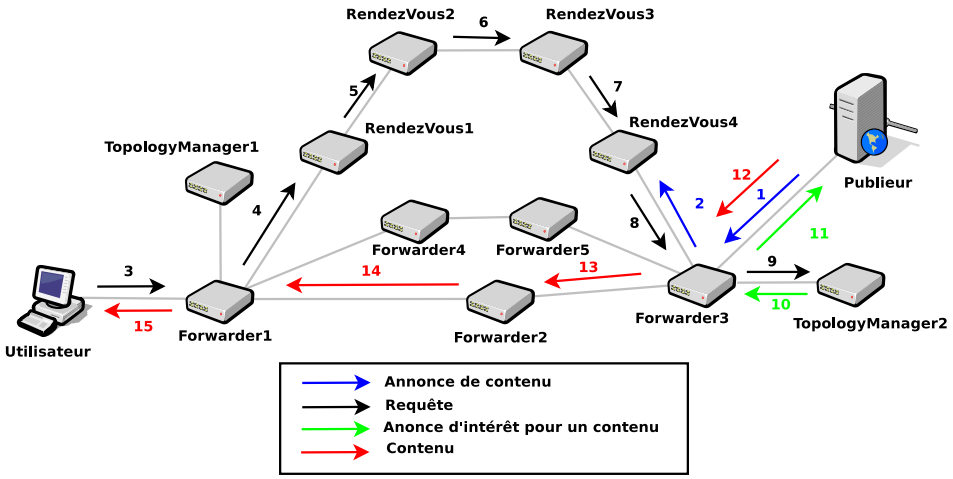


Figure 2.14 – Architecture PURSUIT

2.6.4 *Network of Information* (NetInf)

Développée par le projet SAIL (*Scalable and Adaptive Internet Solution*) [60] composé d'opérateurs, vendeurs et chercheurs qui visent à développer les réseaux du futur, NetInf [22] se veut une architecture réseau qui peut se déployer à l'échelle de l'Internet de manière robuste et fiable. Elle a pour but de pouvoir connecter entre elles plusieurs technologies différentes afin d'être déployées progressivement. Pour ce faire, des éléments des architectures PURSUIT et CCN sont combinés.

NetInf utilise le même nommage que PURSUIT, composé de deux parties. La première correspond à la description de l'entité qui a créé l'objet et utilise un nommage plat, tandis que la seconde utilise un nommage hiérarchique et permet de décrire l'objet.

Elle utilise une organisation à plusieurs niveaux de DHT basée sur des résolveurs globaux des noms. Ces résolveurs permettent d'identifier la première partie du nom (représentant l'entité ayant crée l'objet) afin de savoir où envoyer la requête pour trouver le contenu. La seconde partie du nom (décrivant l'objet de manière hiérarchique) est résolue par un résolveur local, qui permet de trouver quel contenu est demandé. Les résolveurs globaux agissent comme des routeurs IP, acheminant la requête dans le réseau approprié. De plus, chaque nœud du réseau possède un cache permettant de stocker les contenus qui transitent.

Le mode de fonctionnement de NetInf est présenté en Figure 2.15. Le publieur annonce les contenus qu'il possède à son résolveur local de noms (message 1) qui le transmet au résolveur global (message 2). Afin de connaître le chemin pour accéder à un contenu, le souscripteur envoie un message à son résolveur local de noms (message 3). Ce résolveur local va ensuite contacter le résolveur global afin de trouver le publieur (message 4). Le résolveur global retourne les informations sur la localisation du contenu à l'utilisateur (messages 5 et 6) qui va ensuite pouvoir émettre sa requête et la propager jusqu'au publieur (messages 7, 8, 9 et 10). Une fois la requête reçue, le publieur répond au message en envoyant le contenu désiré (messages 11, 12, 13 et 14).

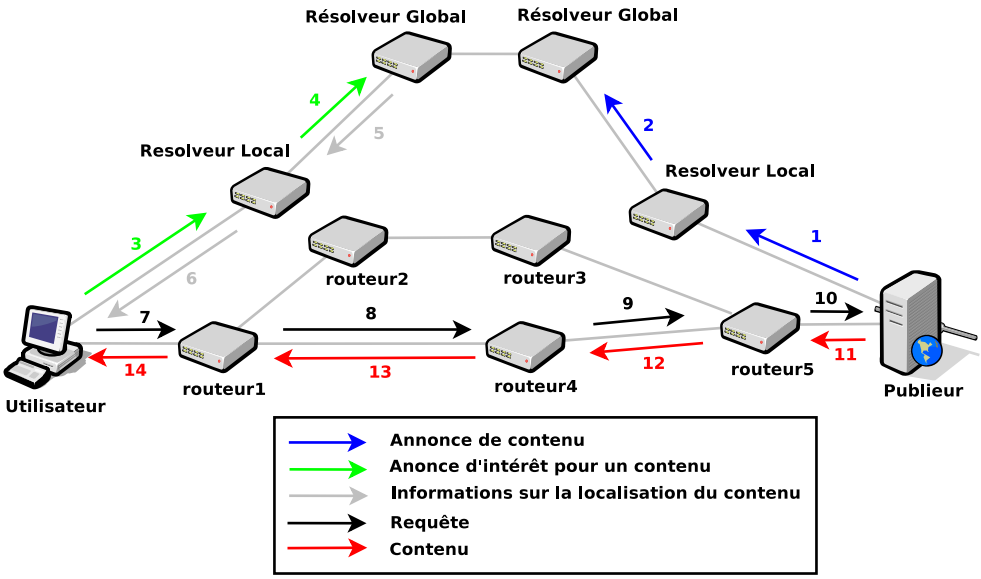


Figure 2.15 – Architecture NetInf

2.6.5 Autres architectures

Les architectures CCN/NDN, PURSUIT, DONA et NetInf sont les principales architectures orientées contenu et elles ont toutes été proposées dans le cadre d'un projet de recherche financé par des agences de financement comme la NSF ou, les Programmes de Recherche nationaux (ANR) ou de l'Union Européenne (FP7, H2020, etc.). Il existe cependant d'autres architectures qui nous ont semblé moins importantes dans la conception du nouveau paradigme des architectures orientées contenu, mais que nous avons souhaité mentionner brièvement dans cette partie.

Par exemple, l'architecture GreenICN [61] du projet Japonais-Européen du même nom, qui repose sur l'architecture NDN et que les auteurs proposèrent de modifier afin de s'intéresser à la problématique de la consommation d'énergie. En effet, avec le déploiement d'un grand nombre de mémoires cache, les architectures ICN peuvent être amenées à consommer une très grande quantité d'énergie. Le but du projet est d'exploiter au mieux une infrastructure capable de fonctionner dans deux exemples de scénarios : le premier est de distribuer efficacement des notifications lors d'une catastrophe telle qu'un ouragan ou un tsunami, lorsque les ressources en énergie et en communication sont rares ; le second est de permettre de diffuser la vidéo de manière efficace et en passant à l'échelle, tant en situation normale que dégradée.

Un autre exemple d'architecture est COMET (*COntent Mediator architecture for content-aware nETworks*) [62]. Issue du projet européen FP7, ayant proposé PSIRP et PURSUIT, cette architecture partage les mêmes concepts de résolution de noms et de routage que l'architecture DONA. L'architecture COMET se divise en deux plans distincts : le plan de médiation de contenus (CMP) et le plan de *forwarding* des contenus (CFP). Le CMP permet de faire le lien entre le contenu et la couche physique, tandis que le CFP permet de gérer la résolution de noms. COMET présente deux modes de fonctionnement, qui permettent de coupler ou découpler la résolution de noms et le *forwarding* de contenus.

2.6.6 Comparaison des architectures orientées contenu

Les architectures présentées ont toutes le même objectif : permettre la distribution de contenu massive à l'échelle de l'Internet en faisant table rase de l'infrastructure Internet existante (*from scratch*). Elles partagent ainsi des concepts et objectifs mais diffèrent également sur les mécanismes leur permettant de fonctionner. Ces derniers sont listés dans la Table 2.1 et discutés ci-dessous :

— **Nommage des données**
Le nommage des données dans les architectures ICN peut être classé en trois catégories : le nommage hiérarchique, le nommage plat, et une combinaison de ces deux derniers (nommage hybride).
Avec le nommage plat, chaque donnée est directement identifiée par un nom ou par le résultat d'une fonction de hachage qui renvoie un identifiant unique du contenu.
Les architectures comme DONA ou PURSUIT utilisent ce type de nommage qui consiste à calculer le nom du contenu à l'aide de fonctions déterministes. Ces fonctions doivent assurer que chaque nœud du réseau obtiendra la même valeur après utilisation de la fonction sur un même contenu, ce qui rend indépendante la localisation du contenu avec son nom de par la structure plate. Ce type de nommage ne permet pas d'être agrégé et pose donc le problème de passage à l'échelle des résolveurs de noms de ces architectures car ils doivent conserver l'ensemble des noms. S'ils sont issus de fonctions de hachage, ces noms ne véhiculent aucune sémantique pour un utilisateur qui ne pourra les mémoriser.

Le nommage hiérarchique quant à lui, est séparé en plusieurs parties permettant de localiser l'objet dans une organisation de noms. Par exemple le nom de contenu `"/video/baseball/Yankees/derek_jeter/walk_off.avi"` définira une vidéo de baseball représentant un joueur du club des Yankees de New Yorks, nommé Derek Jeter, effectuant la frappe permettant à l'équipe de remporter la victoire. Ce mode de nommage est semblable à celui utilisé par les URLs. L'adressage IP est un autre exemple de nommage hiérarchique, dans lequel chaque adresse IP est composée de deux parties, l'identifiant du réseau dans lequel se situe la machine, ainsi que l'identifiant de la machine au sein de ce réseau. On peut aussi citer comme exemple le système de fichier Unix (`/loria/exemple/video/chat.avi`), dans lequel le chemin d'accès nous donne une hiérarchie dans les répertoires, ainsi que leurs localisations.
CCN/NDN utilise un schéma de nommage hiérarchique similaire aux URLs, et dont la sémantique peut être compréhensible par les utilisateurs. Une structure hiérarchique des noms permet d'agréger les noms (comme par exemple par catégorie type /vidéo ou /sports) et ainsi de limiter le nombre de noms à conserver dans les tables des nœuds de l'architecture.

Enfin, une architecture comme NetInf utilise une combinaison de ces deux types de nommage : un nommage plat à l'intérieur des domaines, et hiérarchique entre différents domaines. Le problème de ce type de nommage consiste à repasser d'un nommage plat intra-domaines, à un nommage hiérarchique inter-domaines.

— **Résolution des noms**
La résolution de noms est une fonctionnalité primordiale dans les ICN et elle est liée au nommage des données. Les noms étant indépendants de la localisation du contenu, la résolution de ces noms aura donc pour but de lier une requête à un contenu. Cette liaison permettra par la suite l'acheminement des requêtes vers le ou les nœuds possédant le contenu, ainsi que la transmission des données.
Les architectures utilisant un nommage plat comme DONA, PURSUIT ou NetInf (intra-domaines) nécessitent une entité pour résoudre les noms. En effet, chaque donnée étant nommée de manière unique grâce à une fonction déterministe, les noms ne peuvent pas être agrégés et vont nécessiter un système de résolution pour l'ensemble du réseau.

Avec des architectures possédant un nommage hiérarchique comme CCN/NDN ou NetInf (inter-domaines), les noms peuvent être agrégés et il n'est pas obligatoire d'utiliser une entité globale pour les résoudre, c'est le cas de CCN/NDN. NetInf possède tout de même des résolveurs locaux entre les domaines.

— **Routage des données**
Le routage des données dépend de la résolution de noms, il est donc lui aussi lié à la méthode de nommage des données.
Lorsque la résolution des noms s'effectue par une entité centrale, lors d'un nommage plat, comme cela est le cas avec PURSUIT ou NetInf (intra-domaine), cette entité va calculer la route entre le client et le serveur possédant la donnée. Ainsi, le chemin est mis en place pour la requête.

CCN/NDN, grâce aux noms hiérarchisés, effectue du *forwarding* saut à saut et a pour but de proposer une approche de routage vers le contenu le plus proche (que cela soit une mémoire cache ou un serveur). Les tables de *forwarding* (FIB) se remplissent au gré des données que les nœuds reçoivent et retransmettent. Lors de l'absence de règles pour transmettre une requête, le nœud émet la requête sur l'ensemble de ses autres interfaces et inonde le réseau. Ainsi, CCN/NDN a été conçue sans véritable mécanisme de routage et nous proposerons dans la suite de cette thèse un nouveau protocole de routage adapté à cette architecture.

DONA, bien que possédant un nommage plat, utilise un système de transmission saut à saut se basant sur des résolveurs hiérarchisés. Le résolveur racine du réseau connaît l'ensemble des données à disposition ainsi que le prochain saut pour y accéder. Il agit en quelque sorte comme l'entité globale capable de résoudre les noms, tout en étant un nœud du réseau. Ainsi, l'acheminement des requêtes se fait en remontant vers la racine lorsque le nœud ne connaît pas le prochain saut pour accéder au contenu. Ensuite, le contenu est transmis à l'utilisateur en utilisant le routage IP sur lequel repose DONA.

— **Caching**
La fonctionnalité de mise en cache déployée dans le cœur du réseau est un des nouveaux principes des architectures ICN. Les capacités de stockage vont avoir un impact certain sur l'expérience des utilisateurs ou l'efficacité totale du réseau à transmettre les contenus, qu'elles soient déployées dans l'ensemble de l'infrastructure ou à des endroits stratégiques comme c'est le cas pour les CDNs.
En effet, plus le contenu sera proche de l'utilisateur, plus il y accédera rapidement et verra le temps de réponse diminuer. De plus, maintenir les requêtes locales permettra d'éviter de surcharger le cœur du réseau et de générer de la congestion. Les performances du réseau, mais aussi les accords et partenariats économiques des opérateurs réseaux, seront impactés par la gestion de ces mémoires caches. En stockant les contenus au sein de leurs systèmes autonomes (AS), les opérateurs vont pouvoir réduire le trafic qu'il feront transiter vers les AS voisins, et ainsi diminuer leur coût de fonctionnement.
Actuellement, les architectures ICN permettent aux nœuds de stocker localement les données. Deux cas sont alors possibles : (i) une copie du contenu est récupérée sur le chemin emprunté par la requête vers la source d'origine du contenu, i.e. le serveur (*in-path*) ; (ii) la requête est pro-activement envoyée vers la copie locale la plus proche ; laquelle copie n'est pas nécessairement sur le chemin entre l'émetteur de la requête et le serveur d'origine (*out-path*).
L'architecture PURSUIT ne permet pas de récupérer les données en cache car la requête est acheminée au travers de la chaîne de rendez-vous et seule la réponse passe par le réseau de *forwarder*. Ceci serait possible si les *forwarders* annonçaient les contenus qu'ils gardent en cache au *topology manager* et au *rendez-vous*.
DONA et NetInf vont pouvoir récupérer les données uniquement dans un cache situé sur le chemin entre le client et le serveur (*in-path*). Il serait envisageable de récupérer les données en dehors de ce chemin dans un cache plus proche si ces caches annonçaient les contenus qu'ils possèdent aux résolveurs.
CCN/NDN inondant le réseau avec les requêtes, va pouvoir récupérer les données dans le cache du nœud les possédant le plus proche du client, qu'il soit situé sur le chemin vers le serveur ou non (*in-path* et *out-path*).

Chacune des architectures ICN nécessite donc de pouvoir utiliser un système d'annonce de façon à rediriger les requêtes vers les caches les plus proches possédant la copie demandée, permettant de ce fait un routage dit *anycast*. NDN est la seule solution permettant d'aller chercher la donnée dans un cache, mais ceci est effectué de manière hasardeuse, en inondant le réseau avec les requêtes. En plus du système d'annonces, un système de routage serait nécessaire pour cette architecture.

— **Utilisation du protocole IP**
La seule architecture reposant sur le protocole IP est DONA, afin d'acheminer la donnée du serveur au client. Les autres architectures sont toutes natives et ceci est un point important. Les architectures ICN ont pour but de remplacer l'architecture actuellement en place, il est donc primordial de ne plus se reposer sur le protocole IP et de proposer une nouvelle approche (*clean-slate*).

	Nommage	**Résolution des noms**	**Routage**	**Cache**	**IP/Natif**
DONA	Plat	Hiérarchie de résolveurs	Serveurs	In-path	IP
CCN/NDN	Hiérarchique	–	Inondation	In/Out-path	Natif
PURSUIT	Plat	*Rendez-vous*	Serveurs	Impossible	Natif
NetInf	Plat / Hiérarchique	Résolveurs locaux / globaux	Serveurs	In-path	Natif

Table 2.1 – Comparaison des principales architectures ICN proposées

Choix d'une architecture : Named-Data Networking (NDN)

Pour nos travaux, nous avons choisi de nous intéresser uniquement à l'architecture NDN/CCN. En effet, cette architecture proposée en 2009 est, à ce jour, la plus emblématique et la plus utilisée. Elle repose sur un grand nombre de chercheurs de part le monde et de nombreux projets qui font vivre sa communauté aussi bien de chercheurs académiques que d'industriels. Son principe de fonctionnement, en se basant sur deux messages, est très simple et lui permet de s'affranchir complètement de l'architecture IP.

Au cours de nos travaux de thèse, nous nous sommes intéressés à la problématique de routage qui avait très peu été étudiée jusqu'alors, à l'inverse des mécanismes de gestion de caches [51, 52, 53, 54, 55] ou de contrôle de congestion [63]. En effet, au début de ce doctorat, l'architecture NDN reposait uniquement sur l'inondation du réseau, ce qui ne peut être une solution satisfaisante si l'on souhaite déployer une architecture à large échelle. De plus, l'utilisation d'un réseau de caches permet en théorie d'acheminer la requête vers le cache le plus proche et l'inondation du réseau ne permet pas d'accéder aux caches les plus proches de façon judicieuse.

Ainsi, nous avons proposé dans ce doctorat un protocole de routage original qui permet à NDN/CCN d'acheminer les requêtes vers les contenus sans inonder le réseau, et qui permet également d'accéder à la copie locale la plus proche, c'est à dire de permettre un routage *anycast* dans cette architecture. Dans le chapitre suivant, nous présentons dans un premier temps l'état de l'art concernant les protocoles de routage adaptés à l'architecture NDN qui ont depuis été proposés. Nous présenterons ensuite notre nouveau protocole de routage, ainsi que son évaluation dans les autres chapitres.

2.7 Résumé

Dans ce chapitre, nous avons présenté l'architecture de l'Internet, ainsi que les mécanismes proposés pour lui permettre de relever le défi actuel du transport de contenu à large échelle. En effet, l'Internet est devenu un réseau principalement utilisé pour accéder à du contenu et son modèle de communication de bout en bout n'est pas adapté à ces nouveaux usages.

L'Internet fut alors doté de mécanismes de communication de groupe (*multicast*) mais qui n'ont pu être déployés à son échelle. Les réseaux de distribution de contenu (CDN) ainsi que les réseaux P2P ont aussi permis de distribuer du contenu massivement, mais sans que ces solutions soient optimales pour l'utilisation des ressources du réseau.

Ainsi, plutôt que de tenter d'améliorer l'Internet existant mais vieillissant, de nouvelles architectures réseaux pour un Internet du futur ont été conçues pour être adaptées aux besoins et nouveaux usages (contenu, mais aussi mobilité, sécurité, etc). Ces nouvelles architectures reposent sur un nouveau paradigme de communication orientée contenu (*content-centric*) et place les données au cœur des échanges. Ce sont également des réseaux de cache et les nœuds du réseau peuvent conserver les données pour les retransmettre. Ainsi, les données sont récupérées indépendamment de leurs localisations est c'est un changement de paradigme fondamental avec l'architecture de l'Internet reposant sur TCP/IP.

Parmi ces architectures orientées contenu, nous avons présenté les plus importantes d'entre elles telles que DONA, PURSUIT ou NetInf, mais c'est surtout l'architecture *Named-Data Networking* (NDN) que nous avons particulièrement étudiée et utilisée au cours de ces travaux. En effet, cette architecture est la plus aboutie et bénéficie d'une importante communauté de chercheurs et d'industriels. Nous avons également comparé ces architectures entre elles et discuté leurs mécanismes comme le nommage des données, la résolution des noms, le routage ou l'utilisation des données stockées en cache.

L'architecture NDN ne bénéficiant pas à l'origine de mécanisme de routage permettant d'acheminer les requêtes vers les contenus (au delà du mécanisme d'inondation qui ne peut être déployé à très large échelle), nous avons présenté dans cette thèse de doctorat un nouveau protocole de routage pour l'architecture NDN. Dans le chapitre 3, nous présenterons les autres mécanismes de routage adaptés à NDN. Puis, dans le chapitre 4, nous présenterons notre propre mécanisme de routage SRSC, adapté à NDN, que nous évaluons par simulation dans le chapitre 5, et dont nous déployons une implantation sur notre plateforme d'expérimentation virtuelle (chapitre 6).

Chapitre 3

Routage dans les réseaux orientés contenu

Sommaire

3.1 Introduction 37
3.2 Schémas de *forwarding* 38
3.2.1 Forwarding réactifs 38
3.2.2 Forwarding non-réactifs 41
3.3 Schémas de routage 44
3.3.1 Routage avec signalisation globale dans le réseau 44
3.3.2 Routage avec signalisation locale dans le réseau 49
3.3.3 Routage avec gestion centralisée de la signalisation 51
3.4 Comparaison des différentes solutions 53
3.5 Résumé 54

3.1 Introduction

Nous avons étudié précédemment les principales architectures orientées contenu et discuté de leurs fonctionnements pour pouvoir remplacer l'architecture actuelle de l'Internet. Bien que l'architecture NDN soit largement plébiscitée par les différentes communautés de recherche et l'industrie [46, 64], le déploiement de cette architecture n'est toujours pas d'actualité car elle ne dispose pas de plan de routage adapté.

Le routage est la fonctionnalité qui va permettre aux nœuds du réseau de remplir leur table de transmission, ou de *forwarding*. La fonctionnalité de *forwarding* est celle qui consiste à consulter, pour chaque nœud, sa table de *forwarding* et ensuite à transmettre le message sur l'interface de sortie adéquate, en fonction des informations issues de cette table. Si aucun protocole de routage n'existe ou n'est déployé, les nœuds ne peuvent remplir leur table de *forwarding* et sont alors contraints d'inonder le réseau de messages (*flooding*) en transmettant les messages *Interest* sur toutes les interfaces afin d'atteindre le contenu. Ainsi, au gré des messages *Interest* inondés dans le réseau, les nœuds rempliront leur table de *forwarding* avec de nouvelles entrées.

Cependant, une telle inondation va surcharger le réseau et consommer ses ressources (bande passante, etc.), empêchant un déploiement à l'échelle de l'Internet.

Dans l'architecture NDN, les messages adressent des noms et non plus des hôtes et cela modifie totalement la problématique du routage telle que connue pour l'Internet. Chaque entrée d'une table de *forwarding* va correspondre à un contenu (et non une destination) et indiquer le prochain saut pour y accéder (c'est à dire l'interface de sortie). De plus, les fonctionnalités de cache au cœur du réseau permettent de dupliquer les contenus et de dénombrer plus de copies dans le réseau. Ainsi, une requête vers un contenu pourrait être résolue via plusieurs nœuds pouvant répondre à cette requête. On ne peut donc plus se satisfaire des protocoles de routage utilisés dans l'Internet actuel reposant sur la localisation des hôtes et des communications point-à-point et ce, afin de profiter pleinement de ces nouvelles fonctionnalités de mise en mémoire cache. L'inondation simple du réseau n'est pas non plus acceptable pour pouvoir un jour déployer cette architecture. Il convient donc de définir de nouveaux protocoles de routage adaptés aux spécificités de l'architecture orientée contenu *Named-Data Networking* [65, 66].

Dans ce chapitre, nous présentons les différentes solutions de *forwarding* et de routage proposées pour les réseaux orientés contenu. Toutes ces approches ont bien entendu le même objectif : permettre aux requêtes d'atteindre un contenu et aux données d'être transmises aux utilisateurs et ce, en minimisant le coût dans le réseau. Nous distinguerons par la suite les schémas de *forwarding* au routage. En effet, les schémas de *forwarding* offrent aux nœuds une vision locale, tandis que le routage leur permet d'obtenir une vision globale du réseau et de ses contenus.

3.2 Schémas de *forwarding*

Les schémas de *forwarding* peuvent être réactifs ou non réactifs, selon leur capacité à s'adapter aux modifications intervenant dans le réseau.

3.2.1 Forwarding réactifs

Schéma adaptatif

Dans un réseau NDN traditionnel, la détection de problèmes de transmission sur un lien peut s'avérer longue car les temporisateurs mis en place lors de l'envoi de messages *Interest* sont basés sur le délai aller-retour (RTT) estimé qui est mis à jour lorsque le message *Data* est retourné. Lorsqu'une requête est émise sur une interface d'un nœud, ce dernier ne peut plus transmettre cette requête sur une autre interface avant l'expiration du temporisateur. Les entrées stockées dans la PIT lors de la transmission d'un *Interest*, bloquent toute autre retransmission et ce, même si la donnée n'est pas disponible sur le chemin emprunté.

Pour pallier ce problème, un message *Interest Nack* a été créé dans NDN [67]. Il permet à un nœud ne possédant aucune interface disponible pour émettre une requête, de retourner un message de contrôle au nœud lui ayant envoyé la requête et de le prévenir de l'impossibilité de trouver la donnée sur le chemin emprunté. À la réception de ce message *Nack*, un nœud pourra retirer les entrées de la PIT bloquant la retransmission de messages *Interest* sur une

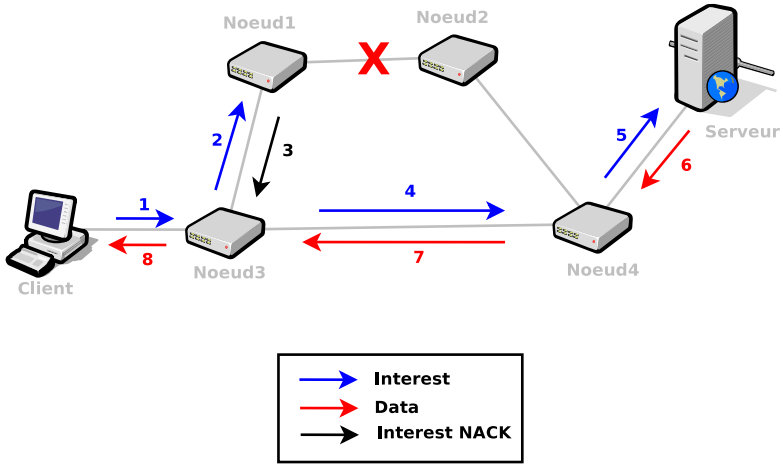


Figure 3.1 – Exemple d'utilisation du schéma de routage adaptatif

autre interface. Le message *Nack* possède un code d'erreur identifiant le problème survenu, tel qu'un lien défaillant ou une donnée qui n'est plus accessible. Les *Nack* sont, en quelque sorte, les messages ICMP de NDN, mais ils sont transmis au nœud précédent et non à la source du message.

Les interfaces sont classées en fonction de la pertinence du chemin pour chaque préfixe, ce qui permet donc de sélectionner l'interface la plus appropriée à chaque requête. Ce classement est mis à jour lors de réception de messages *Nack*, annonçant un problème lors de l'acheminement de la requête, ce qui permet de s'adapter aux conditions du réseau.

Le mode de fonctionnement de ce schéma de transmission adaptatif est présenté sur la Figure 3.1. Sur cet exemple, un client veut accéder à une donnée disponible dans le serveur. Il émet alors un *Interest* pour ce contenu (message 1). Après réception de cet *Interest*, le Noeud3 choisit le meilleur chemin connu pour transmettre la requête et envoie donc ce message vers le Noeud1 (message 2). A la réception du message *Interest*, le Noeud1 ne peut pas émettre le message vers le Noeud2 en raison d'une coupure du lien. Il va alors retourner un *Interest Nack* vers le Noeud3, précisant que le lien est coupé et qu'il ne peut pas accéder au contenu. Le Noeud3 utilise alors un autre chemin, moins performant, mais qui permettra d'atteindre le serveur (message 4 et 5). Suivant les entrées des PITs des nœuds, la réponse du serveur utilisera donc ce second chemin pour acheminer la donnée jusqu'au client (messages 6 à 8).

Bien que permettant de satisfaire les requêtes, même en cas de dysfonctionnement sur le réseau, cette solution peut poser plusieurs problèmes. Premièrement dans un réseau très connecté, le parcours de tous les chemins sera coûteux en cas d'impossibilité de trouver la donnée sur le chemin principal. Deuxièmement, mettre à jour continuellement le classement des interfaces dans chaque nœud et pour chaque contenu, semble une solution très compliquée à mettre en place pour s'adapter à un réseau de grande taille à l'échelle de l'Internet.

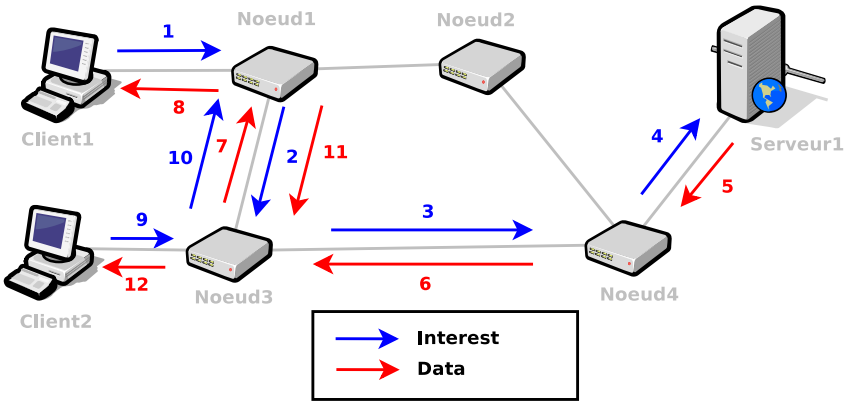


Figure 3.2 – Exemple de fonctionnement du protocole RTR permettant de transmettre les messages *Interest* vers les *Content Stores*

Reverse-Trace Routing Scheme

Afin de maximiser l'utilisation des *Content Stores*, plusieurs propositions de transmission des requêtes sur des chemins multiples ont été proposées.

Le schéma de transmission par chemin inversé ou *Reverse-Trace Routing scheme* (RTR) [68] en est une. Le principe de fonctionnement de RTR consiste à envoyer les requêtes sur les interfaces du nœud ayant reçu auparavant la donnée demandée. Cette méthode vise à augmenter les chances de trouver la donnée dans une mémoire cache, diminuant ainsi la charge du serveur d'origine, mais aussi le nombre de sauts pour l'atteindre.

Pour cela, une nouvelle table de *forwarding* appelée *Network Forwarding Table* (NFT), contenant la liste des messages *Data* ainsi que leurs interfaces d'arrivée dans le nœud est maintenue à jour. Cette table permet de savoir quels chemins ont emprunté les données, de la même manière que la PIT de NDN avec les messages *Interest*, mais ici, pour les messages *Data*.

Lorsqu'un message *Interest* arrive à un nœud, si celui-ci ne possède pas la donnée en cache, il transmet la requête grâce à sa table NFT si une entrée existe (i.e. si un message *Data* pour cette donnée est déjà passé par le nœud), sinon il émet la requête normalement en utilisant les entrées de la FIB.

Le fonctionnement de RTR, présenté en Figure 3.2, nous montre la simplicité et l'efficacité de ce dernier pour réduire le nombre de sauts. Dans cet exemple, deux clients souhaitent accéder à une même donnée présente dans le Serveur1.

Le Client1 émet l'*Interest* qui est envoyé vers ce serveur (messages 1 à 4), car aucun nœud ne possède d'entrée pour ce contenu dans sa table NFT. Le serveur répond avec un message *Data*, qui sera stocké en cache dans le Noeud1 (message 5 à 8). Lors de son passage, les nœuds inscrivent le préfixe de la donnée et l'interface sur laquelle elle a été émise dans leur table NFT.

Lorsque le Client2 émet sa requête (message 9), le Noeud3 voit dans sa table NFT que la donnée a transité récemment par lui et transmet donc le message *Interest* sur l'interface où il a précédemment envoyé le message *Data*, soit vers le Noeud1 (message 10). La donnée trouvée dans le cache du Noeud1 peut donc être retournée vers le Client2 (messages 11 et 12). Si la donnée n'était pas présente en cache, la requête aurait été acheminée vers le Serveur1.

Cependant, le schéma de *forwarding* proposé par RTR se limite à trouver la donnée dans le cache le plus proche, à condition que cette donnée ait déjà transité par ce nœud. Si ce n'est pas le cas, le protocole ne pourra pas trouver la plus proche copie du contenu. De plus, stocker dans chaque nœud les entrées des données qui ont déjà transité peut ajouter de gros problèmes de latence dans un réseau très chargé, car des accès mémoire à répétition vont être engendrés.

Deux autres schémas de *forwarding* utilisant un concept similaire à celui de RTR ont été proposés.

Le premier, nommé *Opportunistic off-path content discovery*, introduit dans chaque nœud NDN une nouvelle table de *forwarding* appelée *Downstream FIB* (DFIB). Celle-ci garde en mémoire la direction empruntée par les messages *Data*, de la même manière que la table NFT de RTR. La différence avec RTR est que dans ce schéma, les messages *Interest* sont transmis en utilisant à la fois la FIB et la DFIB pour assurer un délai minimal d'obtention de la donnée. De plus, un compteur pour chaque entrée de la DFIB permet de connaître le nombre de fois que la donnée a été émise sur le lien. Ceci permet de générer des stratégies de *forwarding* demandant un nombre minimum d'envois pour utiliser le chemin. Ces stratégies ont pour but de réduire le nombre de retransmissions et augmentent la probabilité de trouver le contenu dans une mémoire cache. Les messages *Interest* se voient aussi ajouter un bit supplémentaire pour distinguer les messages suivant les entrées de la FIB de ceux suivant celles de la DFIB.

Le second, appelé *Breadcrumbs*, garde dans chaque nœud des informations sur la dernière interface empruntée par un message *Data* dans le nœud, ainsi que le moment à laquelle celle-ci a été transmise. Pour cela, les nœuds vont stocker des quintuplets contenant : le nom de la donnée, le nœud d'où provient la donnée, le nœud où elle a été transmise, le dernier moment où la donnée est passée par le nœud, ainsi que le dernier moment où une requête pour cette donnée est arrivée au nœud. Grâce à ces informations, les nœuds vont pouvoir émettre les requêtes vers le serveur d'origine, ou bien vers les chemins déjà empruntés par la donnée, en utilisant différentes politiques de routage.

3.2.2 Forwarding non-réactifs

Hash routing scheme

Une méthode de *forwarding* utilisant les fonctions de hachage a été proposée pour les architectures ICN [69]. Cette solution permet de transmettre les requêtes dans le réseau sans avoir à maintenir d'informations dans les nœuds à propos des contenus disponibles. Ce schéma propose de faire le lien entre la mise en cache des contenus, l'acheminement des requêtes et une fonction de hachage déterministe ; fonction qui déterminera dans quel nœud stocker une copie de la donnée en cache, ainsi que le prochain saut à emprunter pour la retrouver.

Le fonctionnement est le suivant : lorsqu'un nœud reçoit une requête, il calcule le résultat de la fonction de hachage sur l'identifiant du contenu demandé. Le nombre résultant de cette fonction détermine l'identifiant du nœud qui peut posséder cette donnée en cache. La requête est alors transmise vers ce nœud. Si le contenu est trouvé dans le *Content Store* du nœud, il est retourné à l'expéditeur de la requête, dans le cas contraire, la requête est acheminée vers le serveur d'origine.

De la même manière, lorsque les contenus sont retournés vers l'utilisateur, ils peuvent être redirigés vers le nœud qui possède le même identifiant que le résultat de la fonction de hachage appliquée à l'identifiant du contenu, dans le but de garder une copie locale en cache.

Déterminer par une fonction de hachage le nœud où stocker une copie locale du contenu réduit la duplication des contenus, tout en évitant les problèmes dus à la coordination des caches. En effet, la fonction de hachage distribuée permet d'obtenir pour chaque contenu, un seul identifiant de nœud.

Cette solution de *forwarding* propose cinq modes différents :

— Symétrique : le chemin d'envoi de la requête et celui de retour de la réponse sont les mêmes. Ils passent par le nœud possédant le même identifiant que le résultat de la fonction de hachage appliquée sur l'identifiant du contenu.

— Asymétrique : la requête est envoyée vers le nœud pouvant posséder le contenu en cache avant d'être transmise au serveur d'origine. La réponse, quant à elle, est retournée en utilisant le plus court chemin. Si le nœud possédant le même identifiant que le résultat de la fonction de hachage appliquée à l'identifiant du contenu n'est pas sur ce chemin, alors la donnée ne peut pas être stockée en cache.

— Multicast : la requête est toujours envoyée vers le nœud pouvant posséder le contenu en cache avant d'être transmise au serveur d'origine. Si la réponse est fournie par le serveur d'origine, elle est dupliquée et un message passe par le chemin le plus court. L'autre message passe par le nœud qui possède le même identifiant que le résultat de la fonction de hachage appliquée à l'identifiant du contenu, dans le but de stocker une copie locale en cache.

— Hybride Symétrique-Multicast : les nœuds décident s'ils utilisent le schéma Symétrique ou Multicast en fonction de la localisation du cache et du nœud destination.

— Hybride Asymétrique-Multicast : les nœuds décident s'ils utilisent le schéma Asymétrique ou Multicast en fonction de la localisation du cache et du nœud destination.

La Figure 3.3 présente le principe de fonctionnement à travers un exemple utilisant le mode Asymétrique. Le client émet une requête pour un contenu dont le résultat de la fonction de hachage appliquée à son identifiant est le même que l'identifiant du Noeud2. Si la donnée est stockée dans un cache, elle ne pourra être présente que dans ce nœud.

La requête (messages 1 à 5) sera alors transmise sur le chemin passant par le Noeud2. Lorsque la requête arrive au Noeud3, celui-ci calcule le résultat de la fonction de hachage appliquée sur l'identifiant du contenu et sait, grâce à un protocole de routage ayant préalablement rempli les FIB des nœuds, que pour atteindre le Nœud2, ayant le même identifiant que le résultat calculé, il doit passer par le Noeud1. Le Noeud1 fait de même et transmet la requête au Noeud2.

Une fois arrivée au Noeud2, la donnée n'étant pas disponible en cache, celui-ci va alors transmettre la requête vers le serveur, qui sera en charge de la traiter. La réponse du serveur va passer par le plus court chemin, car le mode présenté est le mode Asymétrique.

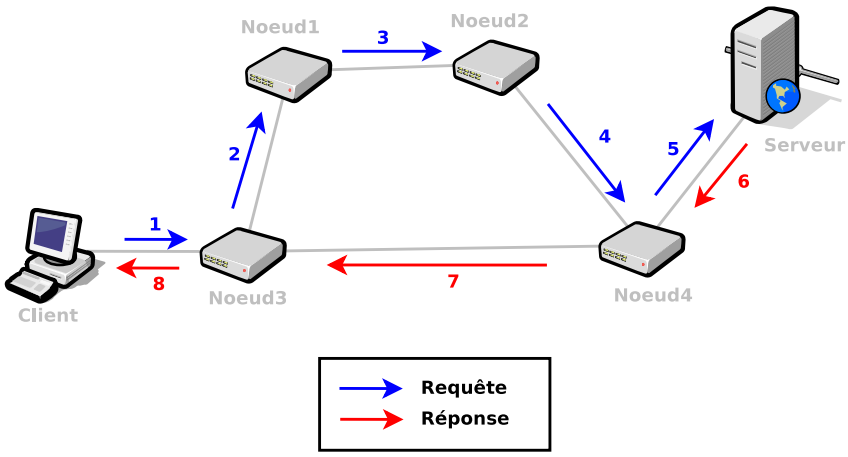


Figure 3.3 – Principe de fonctionnement du schéma de *forwarding* utilisant les fonctions de hachage, en mode Asymétrique ; permettant de retourner la réponse par le plus court chemin

Avec le mode Symétrique, la donnée aurait emprunté le même chemin que la requête et une copie aurait été stockée en cache dans le Noeud2.

Ce schéma permet de maximiser l'utilisation des mémoires cache dans de petits réseaux, mais peut grandement allonger les chemins dans des topologies à l'échelle de l'Internet. Stocker une unique copie des contenus très populaires semble être une solution limitée au vu du nombre de requêtes qui vont pouvoir être émises pour ceux ci. En effet, les requêtes pour un contenu très demandé transiteront toutes par le même chemin, dans le but d'atteindre le nœud possédant le même identifiant que le résultat de la fonction de hachage appliquée sur l'identifiant du contenu. Cet effet va limiter la répartition des charges dans le réseau et va même pouvoir aller jusqu'à l'engorger. Nous pensons qu'il est donc préférable de garder des copies de contenus populaires à plusieurs endroits dans le réseau.

Un protocole de routage sera requis pour remplir les FIB des nœuds dans le but de transmettre les requêtes vers la bonne destination. De plus, lors de la panne ou de l'ajout d'un nœud, les messages ne peuvent plus transiter, à moins de modifier la fonction de hachage dans l'ensemble des nœuds du réseau.

Tous les schémas de *forwarding* présentés permettent d'acheminer les requêtes (messages d'Interest) vers les contenus. Cependant, les schémas de *forwarding* offrent aux nœuds une vision uniquement locale du réseau ; ils peuvent apprendre des entrées mais ne peuvent avoir une vision globale des réseaux ni de tous les contenus à atteindre.

Pour cela, un protocole de routage est nécessaire car il permettra aux nœuds d'obtenir une vision globale du réseau et de ces contenus. Les différents protocoles de routages adaptés à l'architecture NDN sont présentés dans la section suivante.

3.3 Schémas de routage

Nous avons choisi de classer en trois catégories tous les schémas de routage que nous étudierons dans cette section, en fonction des informations de signalisation (*overhead*) qu'ils diffusent dans le réseau :

— (i) Globale lorsque la signalisation est propagée dans l'ensemble du réseau ;

— (ii) Locale quand elle est propre à chaque nœud ;

— (iii) Centralisée si elle est transmise à une entité.

3.3.1 Routage avec signalisation globale dans le réseau

Open Shortest Path First for NDN

Un protocole de routage pour NDN basé sur le protocole OSPF [70] a été proposé. Appelé OSPFN [71], ce dernier utilise le protocole OSPF pour distribuer les annonces de préfixes dans le réseau et calculer les routes vers les serveurs de contenus. OSPF permet de gérer des préfixes identiques venant de plusieurs sites différents grâce à des messages d'avertissement pour chaque donnée accessible dans le réseau.

Tout comme le protocole OSPF, OSPFN fonctionne au sein d'un système autonome (AS), dans lequel chaque routeur maintient une base de données permettant de stocker l'ensemble des préfixes annoncés ainsi que les liens pour y accéder. Cette base de données est mise à jour en continu par l'échange de messages de contrôle entre les nœuds. Chaque nœud de l'AS possédant donc, une fois les mises à jour terminées, la même base de données. Celle-ci permet ensuite au protocole OSPFN de calculer le plus court chemin vers le préfixe souhaité, en décidant à chaque nœud quelle interface utiliser pour transmettre le message.

OSPFN a été implanté dans un réseau CCN au sein duquel chaque nœud possède un démon CCN (CCND) permettant de gérer la FIB et la transmission des messages, un démon OSPF (OSPFD) chargé de gérer le calcul des chemins et une entité locale OSPFN en charge de maintenir à jour la base de données et de faire le lien entre les deux démons.

Les échanges de messages sont représentés sur la Figure 3.4. Sur ce schéma, on voit que les communications entre CCND, OSPFN et OSPFD permettent de mettre à jour la base de données du nœud à travers l'échange de messages de contrôle.

Pour obtenir une route lors de la réception d'un message *Interest* à transmettre, CCND interroge OSPFN, qui à son tour, interroge OSPFD. Ce dernier lui retourne le prochain nœud pour atteindre le serveur d'origine possédant la donnée. OSPFN pourra ensuite mettre à jour les tables de transmission (FIB) de CCND, qui seront ensuite utilisées pour émettre le message *Interest* sur la bonne interface.

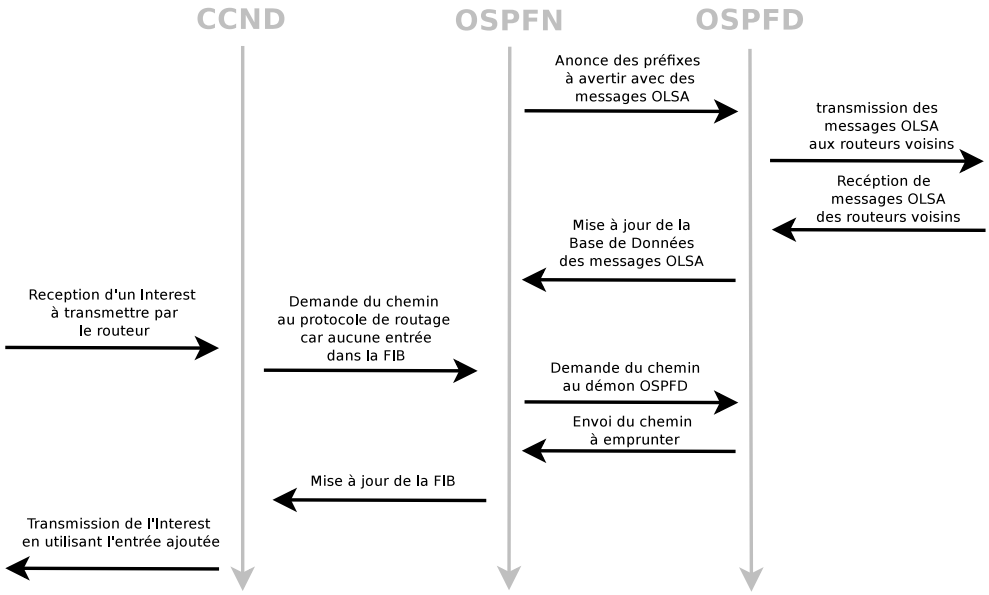


Figure 3.4 – Échanges de messages effectués entre les composants d'un nœuds lors de la transmission d'un *Interest* avec le protocole OSPFN

Le protocole OSPFN ne gérant pas les chemins multiples, il se contente de donner la possibilité à l'opérateur réseau de spécifier une liste ordonnée de futurs sauts qui sera insérée dans la FIB du nœud. Cette liste permet à CCND de sélectionner un autre chemin lorsque la donnée n'est pas retournée via la première entrée choisie.

Un des gros problèmes de OSPFN est sa dépendance avec le protocole IP. À la configuration du réseau, l'opérateur doit mettre en place les interfaces de chaque nœud NDN et faire le lien entre l'adresse IP et le nom de l'interface.

De plus chaque nœud possède une base de données contenant l'ensemble des nœuds du réseau et le prochain saut pour les atteindre, ainsi que l'ensemble des données accessibles dans le réseau. Lors de la modification ou de l'ajout d'un contenu, chaque nœud doit mettre à jour sa table, générant un grand nombre d'échanges de messages de contrôle. Dans un réseau comme l'Internet, où de nouveaux contenus sont mis en ligne continuellement, un nombre très important de modifications est donc généré et propagé, posant de véritables problèmes de passage à l'échelle.

Named-data Link State Routing

Donnant suite à OSPFN, le protocole NLSR [72] (*Named-data Link State Routing protocole*) a vu le jour. Ne se basant plus sur le protocole IP comme son prédécesseur, NLSR utilise les messages *Interest* et *Data* fournis par l'architecture NDN pour envoyer des informations de contrôle. Chaque nœud du réseau est nommé, permettant à NLSR d'utiliser ces noms à la place des adresses IP utilisées dans OPSFN.

Les noms attribués aux nœuds possèdent la structure suivante : `/<reseau>/<site>/<routeur>` Par exemple le routeur du Loria situé à Nancy dans le réseau de l'Université de Lorraine (UL) pourra avoir cette forme : `/UL/Nancy/Loria`.

Chaque nœud maintient à jour la liste de ses voisins grâce à l'échange de messages périodiques. Ces messages ont aussi pour but de détecter les pannes pouvant survenir sur les liens. Dans ce cas NLSR avertit l'ensemble du réseau de ne plus utiliser ce lien. Afin de minimiser les échanges d'informations, chaque nœud transmet le résultat d'une fonction de hachage appliquée sur sa base de données à ses voisins. Ainsi, les nœuds démarrent une synchronisation des bases de données uniquement si ce nombre est différent de celui reçu par leur voisin.

Le protocole NLSR permet d'obtenir plusieurs routes pour chaque préfixe, et n'est plus limité au plus court chemin utilisé dans OSPFN.

Le principe de fonctionnement de NLSR est le même que celui de OSPFN présenté ci-dessus en Figure 3.4. Cependant, NLSR reste un protocole de routage intra-domaine tout comme OSPFN, et le maintien à jour des bases de données dans les nœuds peut rapidement devenir un problème dans un réseau tel que l'Internet où des données sont publiées continuellement.

De plus, le routage vers la source ne permet pas d'optimiser la fonctionnalité de cache proposée par l'architecture NDN. En effet, le protocole permet uniquement de trouver des données dans les caches situés sur le chemin entre la destination et la source. Une donnée disponible dans un cache situé en dehors du plus court chemin reliant la source et la destination ne pourra alors pas être utilisée. La méthode de transmission n'est donc pas *anycast*, comme cela était annoncé initialement pour NDN.
Cependant, NLSR a été désigné comme étant la meilleure alternative actuelle de routage dans NDN, il est donc devenu le protocole de routage par défaut de l'architecture, remplaçant la méthode de *flooding* précédemment utilisée.

Bloom filter-based Routing Approach

Grâce à l'utilisation de filtres de Bloom, BFR [73] (*Bloom Filter-based Routing approach for ICN*) est une solution de routage pour NDN qui ne nécessite pas de découverte de la topologie.

Pour cela, chaque serveur va inonder le réseau avec des messages *Interest* (CAI), annonçant les contenus qu'il met à disposition. Ces messages CAI transportent des filtres de Bloom, afin d'identifier quels sont les contenus disponibles dans le serveur. Ces messages sont stockés dans les nœuds du réseau, ainsi, lorsqu'un nœud reçoit une requête, il va calculer grâce à des fonctions de hachage dans quel filtre apparaît le contenu demandé. Une fois le bon filtre identifié dans l'ensemble des messages CAI que le nœud possède en mémoire, il émet la requête sur les interfaces correspondantes à l'arrivée de ce message CAI.

Le mode de fonctionnement de BFR est schématisé Figure 3.5. Un serveur émet un message CAI contenant le filtre de Bloom qui identifie les contenus qu'il met à disposition (message 1). Les nœuds du réseau gardent ce message en mémoire dans leur PIT, puis le propagent sur l'ensemble des autres interfaces (messages 2 et 3).

Une fois qu'un client émet une requête (message 4), le Nœud 3 calcule, grâce à des fonctions de hachage, dans quel filtre de Bloom se trouve la donnée. Une fois ce filtre trouvé dans l'ensemble des messages CAI disponibles dans sa PIT, il émet le message *Interest* aux prochains sauts possibles pour atteindre le serveur, ici vers le Nœud 4 (message 5). Le Nœud 4 fait de même et transmet la requête au serveur (message 6). La donnée est ensuite retournée vers le client (messages 7 à 9) et peut être stockée en cache pour satisfaire une requête future.

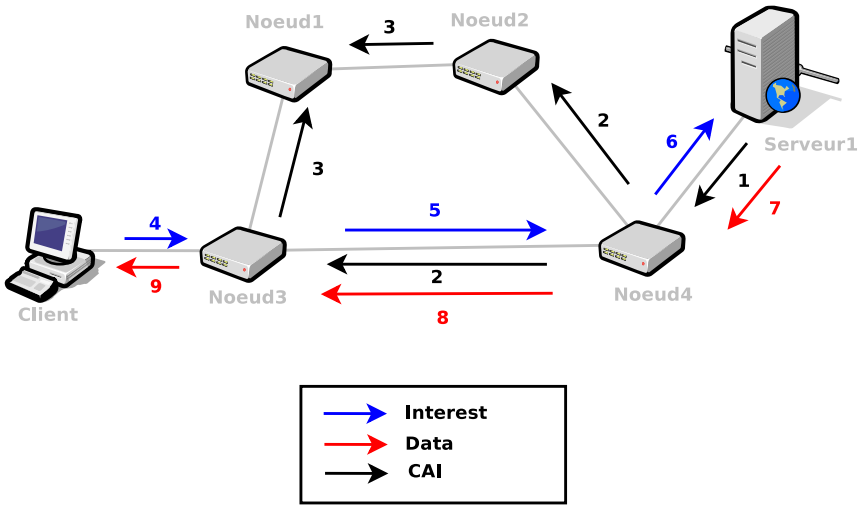


Figure 3.5 – Illustration du mode de fonctionnement de l'approche de routage BFR, basée sur les filtres de Bloom.

Cette solution, bien que ne demandant pas d'échange entre les nœuds une fois l'annonce des contenus effectuée, nécessite une coordination pour s'assurer que l'ensemble des nœuds et des serveurs possède les mêmes fonctions de hachage.

En effet, il est essentiel que chaque nœud du réseau calcule les filtres de Bloom de la même manière. De plus, bien que permettant le *multicast* en utilisant plusieurs chemins pour transmettre les requêtes, la solution de routage *anycast* pour envoyer la requête vers le cache possédant la donnée le plus proche n'est pas implantée. Cette solution ne tire pas pleinement parti du réseau de cache mis à disposition par l'architecture NDN.

Distance-based Content Routing

Une approche de routage de contenus basée sur la distance ou *Distance-based Content Routing* (DCR) [74] permet d'envoyer les requêtes vers le contenu disponible le plus proche. Cette solution ne nécessite pas de maintenir dans les routeurs des informations à propos de la topologie ou de la localisation des contenus.

DCR permet d'envoyer les requêtes vers un seul, plusieurs ou encore tous les sites possédant la donnée recherchée en utilisant un arbre établi par les nœuds du réseau. Les nœuds possédant plusieurs contenus correspondant à un préfixe sont appelés des ancres ou *anchors*. Ces ancres émettent des mises à jour périodiques contenant leurs noms, les préfixes qu'elles annoncent, une distance, ainsi qu'un numéro de séquence.

Chaque nœud possède plusieurs tables pour faire fonctionner DCR. Une table de coût (LT) stocke les coûts des liens vers chaque voisin. Une table des voisins (NT) garde les informations de routage reportées par chacun de ses voisins. Une table de routage (RT) stocke les informations de routage pour chaque préfixe connu du nœud. Et enfin, une table de routage multi-points (MRT) garde les informations de routage des arbres créés nécessitant plusieurs points de communication.

La création des arbres se fait pour chaque ancre, futur sommet de l'arbre. Ces ancres vont émettre un message à leurs voisins directs, précisant leurs noms ainsi que le contenu qu'elles possèdent et une distance nulle. En propageant les mises à jour des ancres saut à saut, chaque nœud met à jour ses tables et incrémente la distance avant de transférer la mise à jour à ses voisins. Ainsi, l'arbre se crée et permet aux nœuds du réseau de connaître à quelle distance ils sont situés des ancres. Il suffit ensuite de choisir vers quelle ancre émettre les requêtes lorsqu'elles arrivent au nœud. Pour cela, le nœud identifie quelle ancre est la plus proche de lui, mettant à disposition le contenu demandé, grâce aux arbres précédemment créés.

La Figure 3.6 permet d'illustrer le fonctionnement de DCR avec deux serveurs placés dans le réseau. Les Noeud1 et Noeud4, liés à ces serveurs, vont donc représenter les deux ancres. Dans un premier temps ces ancres vont s'annoncer au réseau en propageant des messages de mise à jour comme pour l'Ancre2 sur le schéma (message 1 et 2). Ces messages contiennent le préfixe annoncé, le nom de l'ancre, la distance (qui s'incrémentera à chaque retransmission du message) ainsi qu'un numéro de séquence. Dans le cas présenté, le message 1 sera le suivant : <préfixeServeur2, Noeud1, 0, numéroDeSéquence>

Après réception de ce message, le Noeud2 vérifie que le numéro de séquence est inférieur à celui qu'il peut déjà posséder dans sa table de routage pour cette ancre. Si cela est le cas, ou s'il ne possède pas cette entrée, il ajoute dans sa table de routage les informations annonçant que le préfixe du Serveur2 est situé à 1 saut de lui en passant par Noeud1. Il transmet le message de mise à jour à ses voisins (message 2), correspondant à : <préfixeServeur2, Noeud1, 1, numéroSéquence>.

Le Noeud3 ayant aussi reçu la mise à jour de Ancre2 effectue le même processus. Une fois l'arbre terminé, le Noeud4 sait que pour accéder au préfixe du Serveur2, il a le choix d'envoyer les requêtes au Noeud2 ou au Noeud3, les deux chemins étant de même distance. Les nœuds connaissent donc tous les chemins les plus courts vers les préfixes disponibles dans le réseau et utilisent leur table de routage pour transmettre les requêtes vers les ancres.

Dans notre exemple, le client émet une requête pour un contenu localisé sur le Serveur1 (message 3). Le Noeud3, sachant qu'il est situé à 1 saut de l'Ancre1, va émettre la requête directement au Noeud4 (message 4). L'Ancre1 transmet la requête au serveur (message 5), qui va retourner la donnée (message 6). Ensuite, la donnée est acheminée normalement vers le client (messages 7 et 8).

Dans ce schéma, aucune information sur les positions des contenus ou la topologie n'est stockée par les nœuds. Cependant, chaque nœud va devoir conserver sa distance par rapport à chaque ancre, ceci pour chaque préfixe proposé dans le réseau. Lors d'une mise à jour, l'ensemble des nœuds doit alors modifier ses tables, provoquant un potentiel problème de passage à l'échelle dans un réseau aux dimensions de l'Internet.

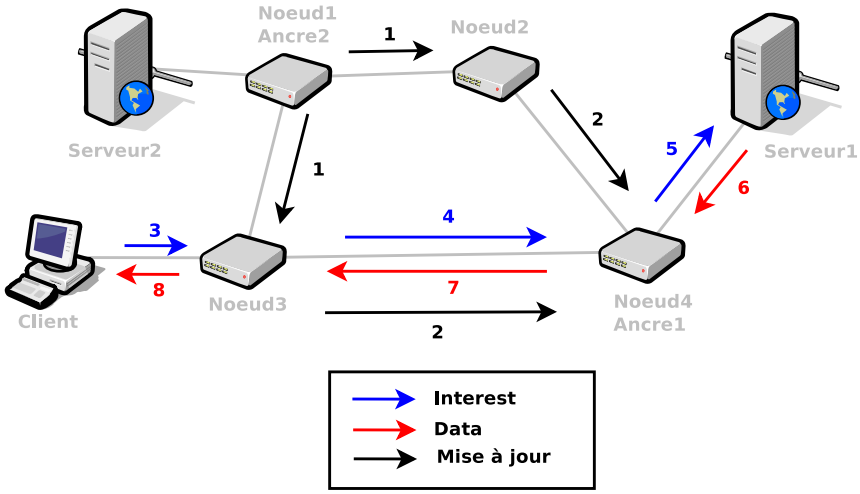


Figure 3.6 – Illustration du mode de fonctionnement du schéma de routage DCR, utilisant la distance pour émettre les requêtes.

3.3.2 Routage avec signalisation locale dans le réseau

Scalable Content Routing for Content-Aware Networking

Le schéma de routage nommé "*Scalable Content Routing for Content-Aware Networking*" (SCAN) [75] utilise simultanément toutes les plus proches copies d'un contenu dans le but de rendre le transfert des données plus efficace.

Avec SCAN, les routeurs suivent le routage traditionnel vers le serveur d'origine fourni par le protocole IP pour garantir une réponse à la requête. Ils utilisent en plus deux tables de routage spécifiques : la table des contenus locaux (LCT) listant les contenus stockés en cache dans le nœud, ainsi que la table de routage des contenus (CRT) qui permet d'obtenir les noms des contenus stockés dans les nœuds voisins. Lors de la réception d'une requête, le routeur va dans un premier temps émettre la requête vers le serveur en utilisant le protocole IP. Il va ensuite vérifier dans sa LCT s'il dispose du contenu demandé, puis dans sa CRT si l'un de ses voisins le possède, de sorte à émettre la requête vers celui-ci.

Afin de limiter les temps de calcul et les délais de réception, les nœuds utilisent les LCT et CRT uniquement si la charge de trafic est faible.

Le mode de fonctionnement de SCAN est illustré en Figure 3.7. Le client souhaite accéder à une donnée présente dans le Serveur1, ainsi que dans le cache du Noeud2. Le client émet donc sa requête, qui est envoyée vers le serveur par le Noeud3 à l'aide du protocole IP (messages 2 et 3). Comme il n'a pas d'entrée concernant la copie au Noeud2 dans sa CRT, il ne peut pas lui envoyer de message. Le Noeud4 lui, sait que le Noeud2 possède la donnée et va donc émettre une requête vers ce dernier de sorte à avoir une autre source potentielle pour tenter de récupérer la donnée plus rapidement (message 4).

Le Serveur1 et le Noeuds2 répondent alors avec un message contenant la donnée et permettent ainsi de maximiser les chances d'obtenir une réponse rapide (messages 5 à 10). Cette solution peut être utilisée pour un contenu trop important, nécessitant plusieurs messages pour l'acheminer à la manière des échanges pair-à-pair avec Bittorrent.

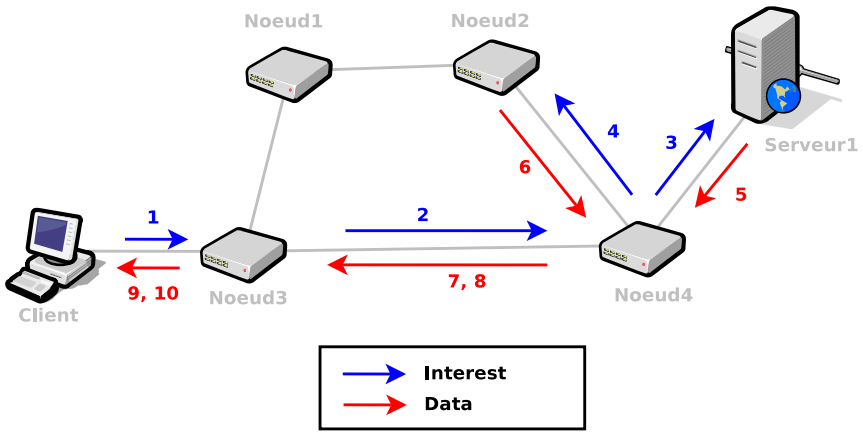


Figure 3.7 – Mode de routage du protocole SCAN permettant l'envoi vers les multiples copies du réseau dans le but d'améliorer le transfert des données.

Bien que proposant une solution intéressante, SCAN repose sur le protocole IP et nécessite d'effectuer le routage vers le serveur dans un premier temps. Nous pensons que ce genre de solution ne tire pas pleinement parti du réseau de cache. En effet, dans une architecture réseau possédant des mémoires cache à chaque nœud, il est important de privilégier l'utilisation des contenus en cache, ainsi que le routage vers le contenu le plus proche et non vers la source, réduisant les délais, le nombre de saut, la charge du réseau ainsi que celle du serveur.

Routage basé sur les capacités de cache disponibles

Un schéma de routage permettant d'utiliser les chemins possédant la plus grande capacité de cache a été réalisé [76]. Pour cela, sur chaque nœud, un *resource manager*, représenté en Figure 3.8, collecte l'historique des requêtes dans le réseau et les capacités de cache. Le *resource manager* analyse les informations sur les différents composants des nœuds puis les configure, comme par exemple la FIB, la PIT et le CS dans les nœuds NDN. Il met en place les chemins pour mener les requêtes jusqu'aux contenus. Il décide aussi des politiques de mise en cache à appliquer dans les nœuds. Les nœuds doivent tout de même communiquer avec leurs voisins afin de ne pas stocker uniquement les mêmes contenus en cache.

Cette méthode peut s'avérer coûteuse et lente car déployer un *resource manager* dans chaque nœud demande un temps de calcul et d'analyse pour chaque saut. De plus, dans des environnements tels que ceux mis à disposition aujourd'hui, où la mobilité est grandissante, cette approche utilisant les statistiques du trafic peut rapidement devenir compliquée à déployer. A chaque changement dans la distribution du trafic, les *resource managers* doivent échanger des informations pour recalculer les chemins, rendant l'opération coûteuse.

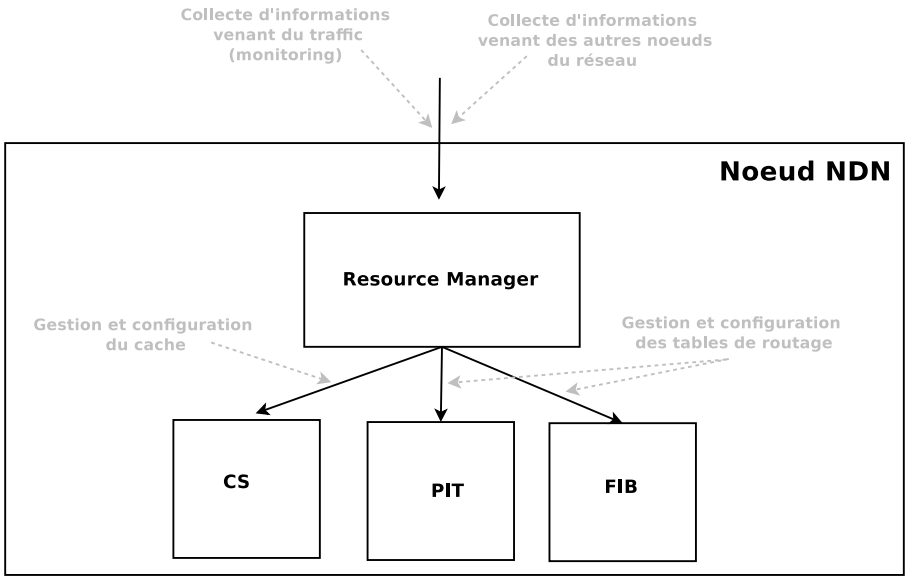


Figure 3.8 – Gestion du nœud NDN par le *resource manager*, permettant la gestion des ressources et la mise en place de politiques de routage spécifiques

3.3.3 Routage avec gestion centralisée de la signalisation

Controller based routing strategy for NDN

Torres et al. proposent CRoS-NDN (*Controller-based Routing Strategy for Named Data Networking*) [77], une solution de routage pour l'architecture NDN basée sur le paradigme des réseaux logiciels.

Une entité, appelée le contrôleur, administre le réseau NDN en mettant à jour les FIB des nœuds, à la demande, lorsque ceux-ci ne possèdent pas de règle pour transmettre une requête. Ce protocole utilise uniquement des messages *Interest* et *Data* pour échanger des informations entre les nœuds et le contrôleur, permettant de ne plus reposer sur l'infrastructure TCP/IP.

Durant une première phase de découverte, le contrôleur prend connaissance de la topologie qu'il gère et chaque nœud identifie ses voisins. Ensuite, lorsqu'un nœud reçoit une requête, s'il ne possède pas de règle dans sa FIB pour la donnée demandée, il contacte le contrôleur pour demander le chemin sur lequel transmettre la requête. Le contrôleur installe alors dans les FIB des nœuds appartenant au chemin calculé, grâce à l'algorithme de Dijkstra, les règles pour acheminer la requête vers le serveur.

Le fonctionnement de CRoS-NDN est présenté dans la Figure 3.9. Les nœuds du réseau sont gérés par un contrôleur, la liaison nœud-contrôleur se faisant lors de la phase de découverte. Le client émet une requête pour un contenu disponible dans le serveur (message 1). Le Nœud3 n'ayant aucune règle dans sa FIB pour transférer la requête, va donc contacter son contrôleur pour lui demander le chemin que la requête doit suivre (message 2).

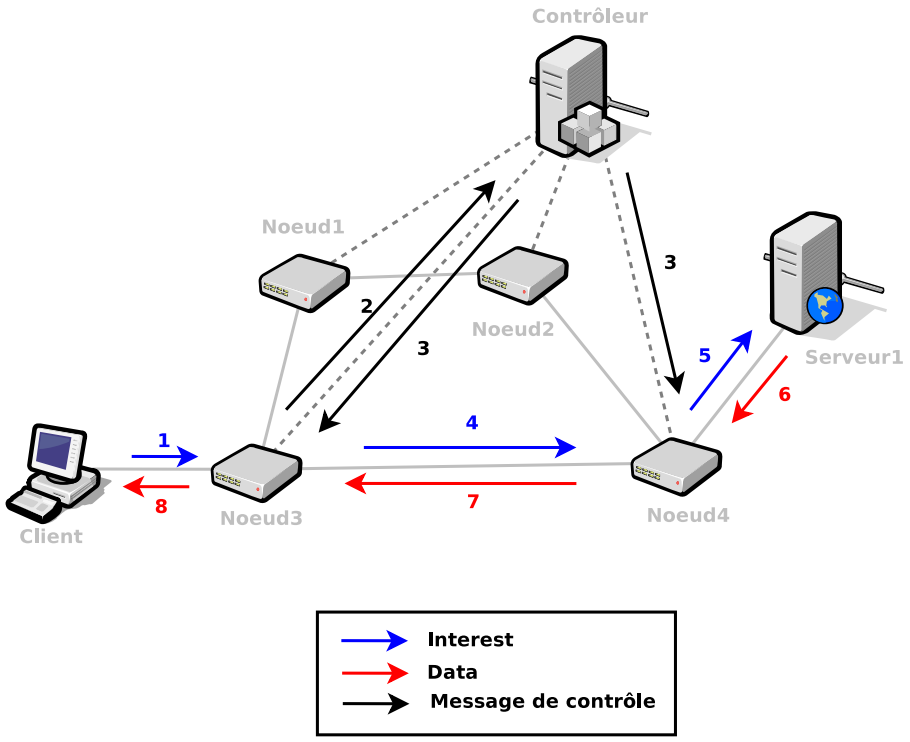


Figure 3.9 – Illustration du mode de fonctionnement du schéma de routage CRoS-NDN, utilisant un contrôleur fourni par le paradigme des réseaux logiciels.

Le contrôleur, grâce à l'algorithme de Dijkstra, connaît le chemin le plus court entre le Nœud3 et le serveur possédant la donnée. Il va donc mettre à jour les FIB des nœuds qui sont sur ce chemin, en ajoutant une règle pour le préfixe demandé (messages 3). Le Nœud3 peut ensuite émettre le message *Interest* sur la bonne interface (message 4) pour utiliser le chemin mis en place par le contrôleur. Le Nœud4, possédant maintenant la règle dans sa FIB, fait de même (message 5) et le serveur répond avec un message *Data* contenant le contenu désiré. Ce message *Data* est retourné au client en empruntant le chemin inverse de la requête, mode de transmission classique de NDN (messages 6, 7 et 8).

CRoS-NDN est proche de la solution que nous présentons dans le chapitre suivant. La différence majeure entre ces deux propositions est la prise en charge du routage *anycast* dans notre approche, permettant d'exploiter au mieux la fonctionnalité de cache dans le réseau. En effet, CRoS-NDN propose uniquement un routage vers le serveur.

De plus, la découverte de la topologie n'est pas réalisée de la même façon. Avec notre solution, seuls les contrôleurs annoncent leurs positions aux nœuds, alors qu'avec CRoS-NDN, chaque nœud inonde le réseau pour s'annoncer aux contrôleurs, générant plus de messages pour cette phase, notamment dans de vastes topologies.

3.4 Comparaison des différentes solutions

Dans cette section, nous proposons un récapitulatif des schémas de *forwarding* et de routage présentés. En effet, utilisant des mécanismes variés, chacun des schémas a ses propres spécificités ou des similitudes qu'il convient de mettre en évidence. Notre synthèse est présentée en Table 3.1 et repose sur les cinq critères suivants :

— **Plan de *forwarding***
Le plan de *forwarding* réactif existe uniquement pour le schéma adaptatif, RTR et DCR. Un tel plan réactif est important pour NDN car les contenus vont être amenés à être dupliqués dans le réseau et il convient d'adapter la transmission à la copie locale la plus proche. De plus, cela permet également de faire face aux éventuelles pannes et autres problèmes qui pourraient survenir dans le réseau.
Les autres schémas de *forwarding* ou de routage sont non réactifs, ce qui veut dire que les nœuds sont incapables de modifier leurs tables de *forwarding* par eux mêmes. Ils ne pourront pas, par exemple, envoyer une requête vers un contenu qui vient d'être stocké en cache à proximité, ou prendre une décision en cas de panne d'un lien.

— **Schéma natif**
Outre OSPFN et SCAN, tous ces schémas de *forwarding* ou de routage peuvent fonctionner directement dans l'architecture ICN associée, sans la pile protocolaire TCP/IP. À nouveau, NDN étant une architecture réseau pour un Internet du futur, il convient de proposer des mécanismes fonctionnant nativement dans ce nouvel environnement pour pouvoir s'affranchir de l'architecture actuelle de l'Internet.

— **Transmission (unicast, multicast, anycast)**
Parmi tous les schémas étudiés, le seul permettant une réelle transmission *anycast*, c'est à dire vers le contenu le plus proche, est DCR.
Toutes les autres solutions transmettent uniquement la requête vers le serveur ou publieur d'origine, c'est-à-dire qu'ils ne permettent pas de récupérer le contenu dans des caches plus proches. Cependant, avec le schéma basé sur la capacité de cache, il est possible de récupérer la donnée dans un cache, si celle-ci est située à un saut du chemin emprunté par la requête. RTR et NLSR permettent une transmission sur plusieurs chemins à la fois (*multicast*), mais pour RTR uniquement sur des chemins répondant à des conditions spécifiques.

— **Signalisation diffusée dans le réseau (overhead)**
Les protocoles OSPFN, NLSR, BFR et DCR nécessitent de propager toutes les mises à jour à tous les nœuds du réseaux. La question du passage à l'échelle de ces protocoles se posera pour de vastes topologies, et notamment à l'échelle de l'Internet. En effet, avec de très nombreux nœuds et contenus, propager toutes les informations de signalisation à tous les nœuds représente une quantité considérable de trafic dans le réseau.
CRoS-NDN permet quant à lui, une gestion de la signalisation centralisée dans une entité de contrôle, ce qui limite la diffusion de ces messages. Les autres schémas (adaptatif, basé sur des fonctions de hachage et la capacité de cache, RTR et SCAN) limitent la signalisation dans le réseau en maintenant les informations localement dans les nœuds.

	Forwarding	**Natif/Non-natif**	**Transmission**	**Signalisation**	**Type**
Schéma adaptatif [67]	Réactif	Natif	*Unicast*	Locales	*Forwarding*
RTR [68, 78, 79]	Réactif	Natif	*Multicast*	Locales	*Forwarding*
Schéma haché [69]	Non-réactif	Natif	*Unicast*	Locales	*Forwarding*
OSPFN [70]	Non-réactif	IP	*Unicast*	Globale	Routage
NLSR [72]	Non-réactif	Natif	*Multicast*	Globale	Routage
BFR [73]	Non-réactif	Natif	*Unicast*	Globale	Routage
DCR [74]	Réactif	Natif	*Anycast*	Globale	Routage
SCAN [75]	Non-réactif	IP	*Unicast*	Locales	Routage
Capacité de cache [76]	Non-réactif	Natif	*Unicast*	Locales	Routage
CRoS-NDN [77]	Non-réactif	Natif	*Unicast*	Centralisées	Routage

Table 3.1 – Comparaison des différents schémas de *forwarding* et de routage pour les ICNs

— **Type (*forwarding*/routage)**
RTR ainsi que le schéma adaptatif et celui basé sur les fonctions de hachage constituent uniquement des schémas de *forwarding*. Pour fonctionner, ils nécessitent également un protocole de routage pour remplir les FIB des nœuds afin de transmettre les requêtes sur l'interface adéquate.

La Table 3.1 illustre bien que, parmi tous les schémas proposés, aucun ne fournit réellement un routage *anycast*. Le seul schéma le permettant est DCR, mais il propage les informations de signalisation de façon globale dans tout le réseau et génère un important trafic de signalisation.

Le protocole NLSR, actuel protocole de routage de l'architecture NDN, présente également l'inconvénient d'une signalisation globale et n'offre pas à proprement parlé un routage de type *anycast*.

3.5 Résumé

Dans ce chapitre, nous avons présenté les principaux schémas de *forwarding* et de routage pour les architectures orientées contenu. Pour les schémas de forwarding, nous avons distingué ceux qui s'adaptent aux changements de condition dans le réseau (schémas réactifs) de ceux qui nécessitent une configuration complète par une entité tierce (schémas non réactifs). Les schémas de routage ont été classés en trois catégories en fonction de la signalisation qu'ils diffusent : globale, locale ou centralisée.

Parmi toutes ces approches, la plupart d'entre elles ne permettent pas le routage *anycast*, c'est à dire qu'elles n'exploite pas le réseau de cache dans l'architecture NDN. Les données sont récupérées en tentant d'interroger le serveur d'origine, mais les requêtes ne sont pas envoyées vers la copie locale la plus proche de l'utilisateur. De plus, les approches à signalisation globale présentent le risque de générer une importante quantité de trafic de signalisation et notamment pour des très grandes topologies à l'échelle de l'Internet.

Dans le chapitre suivant, nous présentons notre protocole de routage, appelé SRSC, adapté à l'architecture NDN. En se basant sur le paradigme SDN, notre protocole permettra un routage de type *anycast*, tout en limitant l'impact du trafic de signalisation dans le réseau.

Chapitre 4

SRSC : Protocole de routage basé sur SDN

Sommaire

4.1 Introduction 57
4.2 Intégration de SDN dans les ICN 58
4.2.1 Intégration SDN-ICN avec IP 58
4.2.2 Intégration SDN-ICN natif 60
4.3 *SDN-based Routing Scheme for CCN/NDN* (SRSC) 62
4.3.1 Principes de fonctionnement 62
4.3.2 Phase d'amorçage (i.e. *Bootstrapping*) : 67
4.3.3 Phase d'acheminement des règles (i.e. *Rules Forwarding*) : 76
4.4 Résumé 82

4.1 Introduction

Nous avons vu précédemment que les protocoles de routage pour les architectures orientées contenu peinaient à bénéficier des fonctionnalités de cache du réseau. En effet, la plupart de ces protocoles ne permettent pas un routage dit *anycast*, c'est-à-dire qu'ils ne redirigent pas les requêtes vers les copies locales les plus proches.

En parallèle de nos travaux, le paradigme des réseaux logiciels ou *Software-Defined Networking* (SDN) [80] est devenu l'un des sujets majeurs de la recherche en réseaux informatiques aussi bien dans la communauté scientifique qu'industriel.

Le paradigme SDN a comme objectif de séparer le plan de données (transmission des paquets) du plan de contrôle (décision de routage) dans le réseau et ce, afin de permettre une certaine flexibilité dans le réseau, là où l'Internet est parfois vu comme un réseau ossifié pour lequel plus aucune modification n'est possible. Pour cela, une entité appelée contrôleur va être en charge de configurer le matériel réseau en fonction de politiques précisées par l'administrateur. Le contrôleur centralise les informations du réseau, comme par exemple la topologie ou les statistiques de trafic, qui vont lui permettre d'adapter les règles de routage. Ainsi, les nœuds

du réseau, c'est-à-dire les routeurs pour l'Internet, deviennent uniquement des entités de transmission des messages (*forwarding*) en fonction des règles établies par le contrôleur. En l'absence de règles pour transmettre un message, le nœud contacte son contrôleur grâce à un protocole de communication, dont une des normes actuelles est le protocole OpenFlow [81]. Le contrôleur indique ensuite au nœud comment traiter le message en lui transmettant les règles adéquates. Cela permet d'effectuer un routage fin dans le réseau, au flux près ou au paquet près et non plus simplement en fonction de l'adresse destination des datagrammes IP.

Dans ce chapitre, nous proposons un protocole de routage, appelé SRSC (*SDN-based Routing Scheme for CCN/NDN*), qui combine le paradigme SDN et l'architecture NDN afin d'utiliser au mieux les fonctionnalités de cache et les multiples copies dans le réseau.

En effet, l'architecture NDN ne dispose pas, à proprement parlé, de plan de contrôle capable de prendre les décisions de routage et les nœuds ne font que retransmettre les messages *Interest* quitte à inonder tout le réseau (*forwarding* uniquement). Le paradigme SDN peut compléter l'architecture NDN en le dotant d'un plan de contrôle pour le routage. De plus, le contrôleur ayant une vision omnisciente du réseau et des contenus, sera à même d'indiquer les caches locaux les plus proches pour acheminer les requêtes et ainsi d'assurer un routage *anycast* vers le contenu. Son approche centralisée permettra également de contenir le trafic de signalisation et de ne pas inonder le réseau de toutes les annonces. Ainsi, pour acheminer un message *Interest*, chaque nœud pourra interroger le contrôleur pour connaître le nœud le plus proche vers lequel transmettre le message.

Avant de décrire plus en détail notre protocole SRSC, nous présentons dans la section suivante les différents travaux proposant d'intégrer le paradigme SDN aux architectures orientées contenu.

4.2 Intégration de SDN dans les ICN

Les travaux visant à intégrer les réseaux logiciels (SDN) dans les architectures ICN peuvent être classés en deux catégories. La première catégorie regroupe les solutions proposant de déployer progressivement les architectures ICN sur l'Internet, en s'intégrant avec le protocole IP. La deuxième catégorie recense les approches voulant s'affranchir de l'architecture IP existante, en modifiant le protocole OpenFlow pour qu'il puisse prendre en compte les messages des architectures ICN.

4.2.1 Intégration SDN-ICN avec IP

Continuer à utiliser le protocole IP permet un déploiement progressif des architectures orientées contenu sans devoir opérer un changement radical de l'infrastructure de l'Internet actuel et de ses équipements.

Deux solutions avec OpenFlow ont été proposées pour améliorer la fonctionnalité de cache dans l'architecture CCN [82, 83]. Pour cela, un adaptateur entre le commutateur OpenFlow et le nœud CCN transforme, grâce à une fonction de hachage, les noms des contenus gérés par CCN en données transportables par OpenFlow. Ces données sont, par exemple, une adresse IP ou un

port. Grâce aux fonctionnalités de supervision du trafic fournies par OpenFlow, le contrôleur a la possibilité d'identifier les contenus populaires. Il pourra donc décider de stocker ces contenus dans les caches dont l'adresse correspond à celle qui résulte de la fonction de hachage. Cela lui permettra par la suite de rediriger les requêtes vers ces contenus.

Ces solutions sont très proches de la solution de *forwarding* utilisant les fonctions de hachage présentée dans le chapitre précédant [69], mais elles centralisent au sein d'un contrôleur SDN, les informations du réseau et la fonction de hachage, ce qui permet de s'adapter plus facilement aux différents changements du réseau.

Toujours dans le but d'intégrer le contrôleur SDN dans les architectures ICN en environnement IP, un serveur intermédiaire, ou proxy, filtre les requêtes HTTP des clients [84]. Ce proxy extrait les contenus demandés des requêtes HTTP et contacte ensuite le contrôleur SDN à l'aide d'OpenFlow pour connaître les emplacements des potentielles répliques de ces contenus. Il redirige ensuite les requêtes en fonction des réponses du contrôleur. Le contrôleur, quant à lui, centralise les informations sur les localisations des contenus et installe les règles de mise en cache dans les nœuds. L'utilisation de filtres de bloom permet de contacter le contrôleur uniquement pour des contenus populaires ; les contenus impopulaires sont directement demandés au serveur d'origine pour limiter les communications inutiles avec le contrôleur.

Dans [85], le contrôleur SDN achemine les requêtes ICN sans connaître le protocole utilisé. Cette solution encapsule les requêtes de l'architecture ICN dans des datagrammes IP. Lorsqu'un nœud du réseau reçoit une requête ICN, il contacte son contrôleur qui peut analyser la requête et ensuite installer les règles dans les nœuds IP jusqu'au nœud ICN possédant la donnée demandée, en attribuant une adresse IP privée au nœud destination. La requête d'origine sera modifiée pour pouvoir la faire transiter sur le réseau IP. Le contrôleur proposé est donc capable de fonctionner à la fois avec le protocole IP et le protocole ICN utilisé.

Chang et al. proposent C-Flow [86], une architecture orientée contenu fonctionnant au dessus du protocole OpenFlow. Afin de permettre à OpenFlow d'être orienté contenu, une adresse IP privée est associée à chaque contenu qui doit être délivré. Le contrôleur SDN peut ainsi garder dans une table la liste de tous les contenus stockés dans les nœuds du réseau. Les requêtes, ainsi que les réponses, sont transmises en utilisant ces adresses privées. Par ailleurs, le protocole OpenFlow est étendu pour prendre en charge les fonctionnalités de mise en cache à l'intérieur du réseau et ainsi annoncer aux nœuds où stocker les contenus.

Cette solution est très proche de la précédente, mais les adresses IP ne sont plus attribuées aux nœuds mais aux contenus. L'un des désavantages de C-Flow est qu'il n'est pas envisageable de proposer une adresse IP privée pour chacun des contenus, car à l'échelle de l'Internet, on dénombrerait beaucoup plus de contenus que d'adresses privées et tous les contenus ne pourraient donc pas être identifiés.

Une autre technique liant des contenus à des adresses privées assignées par le contrôleur SDN permet de réaliser une interconnexion de réseaux de contenus [87], et ainsi de gérer la distribution de contenus entre plusieurs ISPs utilisant OpenFlow. Cette solution présente cependant le même désavantage que C-Flow en matière de pénurie d'adresse par rapport au nombre de contenus.

NDNFlow [88] utilise un contrôleur SDN OpenFlow pour effectuer la transmission de paquets dans un réseau où certains nœuds intègrent le protocole NDN. Pour cela, deux modules NDN spécifiques, un pour le contrôleur et un pour le client, permettent de déployer dans un réseau OpenFlow les fonctionnalités de cache de NDN ainsi que le routage en fonction des préfixes. Lorsqu'un nœud ne possède pas de règle pour transmettre un *Interest*, ce dernier contacte le module NDN du contrôleur. Ce module sera chargé de mettre en place un chemin vers le contenu demandé et de configurer les nœuds utilisant les fonctionnalités NDN. Pour les autres nœuds, le module contrôleur configure les règles pour effectuer un tunnel IP lors du passage de la requête par ces nœuds.

Cette méthode permet à un réseau SDN de gérer simultanément les requêtes IP et NDN en fournissant les deux modules au contrôleur et aux nœuds du réseau. Cependant, l'acheminement de paquets impose un long délai de configuration pour les nœuds.

Veltri et al. [89, 90, 91] intègrent l'architecture ICN CONET dans un réseau OpenFlow. CONET utilise des champs redéfinis du protocole IP pour transporter les données nommées provenant du réseau ICN à travers les routeurs IP ; par exemple les champs nommés *Diffserv* et *Type*, permettant de différencier les diverses qualités de service que le réseau peut fournir, ainsi que le type de contenu transporté par le message.

Plusieurs pistes futures sont envisagées, dont une solution dite à "court terme" visant à adapter les messages ICN pour qu'ils soient pris en compte par le protocole OpenFlow existant. Puis, une solution à "long terme" souhaite étendre OpenFlow en proposant une API spécifique pour les architectures ICN et veut se conformer à la vision centrée sur la donnée et non plus centrée sur les hôtes. Seule l'approche à "court terme" a été pour l'instant implantée.

Tous ces travaux permettent de mettre en œuvre des architectures orientées contenu et plus particulièrement NDN/CCN (ou CCN-like) dans l'environnement IP actuel. Cependant, elles restent un frein au déploiement des architectures ICN car elles reposent toujours sur l'infrastructure IP. En effet, attribuer des adresses IP à des contenus, ou bien encore le résultat du hachage du préfixe du contenu à une adresse IP, sont des solutions qui peuvent difficilement passer à l'échelle d'un réseau comme l'Internet.

D'autres pistes doivent donc être envisagées pour proposer un déploiement et un routage performant dans les architectures ICN, sans le protocole IP, comme nous le verrons à la suite de cette section.

4.2.2 Intégration SDN-ICN natif

Liu et al. [92] proposent une architecture unifiant les ICN grâce à SDN. Elle repose sur trois composants pour communiquer : les programmes de contrôle, le contrôleur et les commutateurs classiques ou d'accès. Les commutateurs classiques acheminent les paquets alors que les commutateurs d'accès encapsulent les paquets ICN à l'aide de règles établies par le contrôleur. Ces commutateurs d'accès servent à faire le lien entre les différents réseaux ICN déployés. Lorsqu'un paquet ne peut pas être transmis par un commutateur, il est envoyé à son contrôleur, qui le

retransmet au programme de contrôle correspondant à son protocole ICN afin qu'il émette la requête dans le réseau. Pour cela, le contrôleur va établir les chemins dans les différents nœuds traversés, en fonction du protocole ICN utilisé.

Cette solution permet de faire cohabiter plusieurs architectures ICN en s'affranchissant du protocole IP et en tenant compte uniquement des propriétés et des performances de l'architecture ICN sous-jacente. Mais il n'y a, à ce jour, à notre connaissance, aucune mise en œuvre de cette solution complexe.

Pour optimiser le comportement du réseau, Chanda et al. [93] extraient des informations sur les données qui y transitent. Ils exploitent pour cela le plan de contrôle fourni par le paradigme SDN. Ces meta-données permettent d'effectuer de l'ingénierie sur le trafic, d'améliorer la gestion des caches et de proposer un service de pare-feu pour les réseaux ICN.

Le contrôleur SDN communique avec les nœuds via le protocole OpenFlow qui a été modifié pour prendre en compte des actions du type extraction de méta-données dans les messages ICN. L'extraction de données peut se faire au niveau de la couche réseau ou applicative. Pour l'extraction au niveau de la couche réseau, les tailles des contenus sont identifiées et classées en deux catégories : petite ou grande taille. Au niveau de l'application, les en-têtes HTTP sont analysées. Il est possible de privilégier, soit la complexité de l'extraction des méta-données, soit la rapidité des actions à mettre en place (pare-feu, mise en cache, calcul de chemins, transfert de charge).

Cette solution ne règle cependant pas le problème du routage mais permet de bénéficier de la centralisation des données et du plan de contrôle dans le réseau.

Sysivelis et al. [94] montrent l'intérêt de combiner les technologies émergentes que sont les réseaux SDN et les ICN. Ils proposent une plateforme expérimentale (*testbed*), reposant sur l'architecture ICN nommée *Blackadder*. L'objectif est de séparer chaque fonctionnalité du nœud ICN, comme la gestion de la topologie ou bien la création de rendez-vous. Grâce à l'utilisation du protocole Openflow, les en-têtes générés par les nœuds Blackadder ne nécessitent plus d'en-têtes Ethernet, ce qui permet de réduire la surcharge inutile du réseau. La fonction de transmission de messages de Blackadder a été remplacée par celle fournie par le contrôleur SDN et donne ainsi la possibilité de transmettre des flux entrants avec des labels inconnus.

Cependant, cette solution ne prend pas en compte la proximité des caches pour acheminer les données.

Le système de service de chaînage des fonctions, ou *Function-Centric Service Chaining* (FCSC) [95], repose sur le paradigme orienté contenu des ICN afin de proposer une solution de gestion des réseaux SDN. Il utilise des fonctions virtualisées, nommées, et accessibles par ces noms et non plus par leurs localisations. Le nommage des fonctions est réalisé à partir de préfixes similaires au nommage des contenus de l'architecture NDN, dans le but de répondre aux problèmes de flexibilité, de dynamicité et de fiabilité des solutions SDN. En effet, une fonction peut être demandée à l'ensemble des instances la possédant et non plus par sa localisation.

Avec ce système de nommage, le contrôleur SDN ou la source émettrice du paquet peuvent décider d'appliquer des fonctions spéciales (inspection de paquets (DPI), mise en cache, ...).

Le paquet est encapsulé avec un en-tête représentant la séquence des actions à effectuer. Ici, le réseau NDN n'achemine plus du contenu, mais le plan de contrôle des paquets. Ce système ne constitue pas réellement une intégration de SDN dans les ICN, mais plutôt une utilisation des concepts des architectures ICN dans les réseaux logiciels (SDN).

Nous venons de présenter différentes intégrations de solutions SDN dans les architectures ICN. Ces études montrent qu'il est possible de faire cohabiter le paradigme SDN au sein de cette nouvelle architecture pour gagner en flexibilité pour le routage et la gestion des informations dans le réseau.

Dans la section suivante, nous détaillons notre protocole de routage SRSC, qui combine le paradigme SDN avec l'architecture NDN, pour tirer parti des caches du réseau et permettre un routage *anycast*.

4.3 *SDN-based Routing Scheme for CCN/NDN* (SRSC)

SRSC (*SDN-based Routing Scheme for Content-centric networks*) est un protocole de routage pour NDN intégrant le paradigme SDN. Notre solution ne repose plus sur l'architecture TCP/IP de l'Internet (*clean-slate approach*) car nous souhaitons permettre le déploiement de l'architecture NDN. Une dépendance à l'environnement IP freinerait son déploiement effectif et pourrait, à terme, remettre en cause son réel déploiement à l'instar des avancées tels que le *multicast* réseau, les architectures d'intégration de service, ou d'une certaine façon IPv6.

Notre protocole SRSC utilise uniquement les messages définis par l'architecture NDN, à savoir les messages *Interest* et *Data*, pour gérer les communications entre le contrôleur et les nœuds. Dans cette section, nous abordons tout d'abord les grands principes de fonctionnement du protocole. Nous présentons ensuite les différents préfixes des messages *Interest* et *Data* du protocole SRSC. Les algorithmes du protocole sont ensuite expliqués et illustrés à travers un exemple.

4.3.1 Principes de fonctionnement

Le protocole SRSC exploite le paradigme *Software-Defined Networking* pour transmettre des règles de routage de façon réactive dans le réseau. En effet, les contenus étant propagés grâce aux fonctionnalités de cache dans le cœur du réseau NDN, le but est donc d'apporter une solution de routage *anycast* pour que les requêtes soient envoyées vers le contenu le plus proche de l'utilisateur et non systématiquement sur un chemin vers le serveur d'origine.

Entité de contrôle

Un contrôleur SDN permet de collecter les informations sur le réseau, comme par exemple, la topologie et la localisation des contenus. Le contrôleur SRSC va donc être en charge du plan de contrôle du réseau comme indiqué sur la Figure 4.1 ; les nœuds du réseau se consacrent uniquement au plan de données (*forwarding*). Il va devoir communiquer avec les nœuds qu'il administre dans le réseau afin d'échanger des informations et établir les chemins jusqu'au contenu pour acheminer les requêtes.

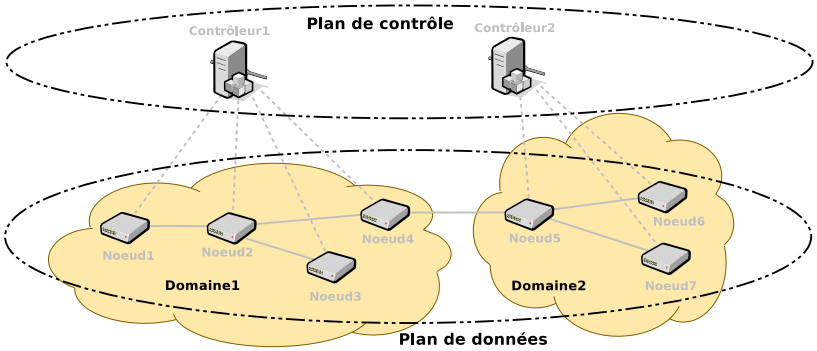


Figure 4.1 – Plan de routage SDN pour architecture NDN.

Habituellement, ces communications se font grâce à un protocole de communication, comme OpenFlow. Nous proposons une nouvelle approche où le canal de communication "nœud-contrôleur" sera composé uniquement de messages *Interest* et *Data*, de sorte à ce qu'ils s'intègrent parfaitement dans l'environnement NDN, sans avoir à ajouter de niveau d'abstraction supplémentaire, proxys, ou APIs, etc. Ces messages utiliseront des préfixes spécifiques que nous détaillerons dans la suite de ce chapitre.

Phases de SRSC

SRSC fonctionne en deux phases distinctes. La première phase, appelée amorçage ou *Bootstrapping*, permet au contrôleur d'apprendre la configuration du réseau dont il a la responsabilité (nœuds, topologie, contenus, etc.). La seconde phase, nommée *Rules Forwarding*, consiste à la mise en place, par le contrôleur, des règles de transmission dans les nœuds du réseau.

Au cours de ces deux phases, le contrôleur joue ainsi plusieurs rôles. Durant le *Bootstrapping*, il va devoir construire la topologie de son réseau et identifier les nœuds de bordure, qui sont les nœuds servant de passerelles entre les différents domaines des contrôleurs. Il sera informé par les nœuds des contenus disponibles dans le réseau pour pouvoir ensuite diriger les requêtes vers le contenu le plus proche et choisir les règles de transmission pour les nœuds NDN.

Dans la phase de *Rules Forwarding*, le contrôleur continue de recevoir des annonces de contenus disponibles sur les caches au gré des échanges dans le réseau et des stockages sur les nœuds. En ayant obtenu par la phase de *Bootstrapping* la connaissance complète du réseau, de sa topologie, des nœuds et des contenus qu'ils possèdent et en étant périodiquement mis à jour de la localisation des nouveaux contenus dans le réseau ou des éventuels changements du réseau, le contrôleur va pouvoir calculer les chemins entre les différents nœuds pour déterminer la route qu'une requête doit emprunter. Pour cela, lorsqu'un nœud souhaite connaître le chemin pour accéder à un contenu, le contrôleur détermine grâce à l'algorithme de Dijkstra [27] le plus court chemin entre le nœud en question et chaque nœud possédant la donnée (ou les nœuds de bordure si la donnée n'est pas disponible dans ce réseau). Une fois calculés, les chemins sont stockés dans le contrôleur, limitant ainsi le nombre d'appels à l'algorithme de Dijkstra.

L'objectif de notre protocole est donc de maintenir pour le contrôleur sa vision omnisciente du réseau qu'il administre afin qu'il puisse transmettre aux nœuds les règles vers le contenu stocké sur le nœud le plus proche.

Structures de données du Contrôleur

Dans SRSC, plusieurs structures de données sont utilisées au sein du contrôleur.

La topologie est stockée dans une table, où la première entrée représente les nœuds du domaine, la seconde représente la liste des voisins associée au coût du lien entre chacun d'eux. Les coûts de chaque lien permettront de définir dans le contrôleur des politiques de routage spécifiques. L'exemple décrit dans la Table 4.1 représente la topologie stockée par le Contrôleur1 de la Figure 4.1.

Noeud	**Liste des voisins**
Nœud1	< (Nœud2, 1)>
Nœud2	< (Nœud1, 1) ; (Nœud3, 1) ; (Nœud4, 1)>
Nœud3	< (Nœud2, 1)>
Nœud4	< (Nœud2, 1) ; (Nœud5, 1)>

Table 4.1 – Exemple de table de topologie présente dans les contrôleurs SDN. Ici le Contrôleur1 de la Figure 4.1 possède quatre nœuds, reliés par des liens de même coût (1 dans cet exemple)

Les nœuds de bordure sont stockés dans une table contenant, pour chaque identifiant de nœud de bordure du domaine, l'identifiant du contrôleur auquel il est lié. Cette table permet de différencier tous les domaines voisins et d'identifier les passerelles pour y accéder. Par exemple, dans la Figure 4.1, le Contrôleur1 possédera la Table 4.2.

Nœud de bordure	**Contrôleur du domaine**
Nœud5	Contrôleur 2

Table 4.2 – Exemple de table de bordure présente dans les contrôleurs SDN. Ici le Contrôleur1 de la Figure 4.1 possède un seul nœud de bordure permettant de joindre un autre domaine, le Nœud5 lié au Contrôleur2.

Une table de contenus permet de stocker, pour chaque contenu présent dans le domaine, l'ensemble des nœuds qui en détiennent une copie. À chaque préfixe est donc associée une liste de nœuds qui possèdent ce contenu. Un exemple est présenté en Table 4.3.

Contenu	**Noeuds**
`/loria/exemple/video/chat.avi`	<Nœud2, Nœud4>
`/video/baseball/Yankees/derek_jeter/walk_off.avi`	<Nœud3>

Table 4.3 – Exemple de table de contenus présente dans les contrôleurs SDN. Ici le Contrôleur1 de la Figure 4.1 possède deux contenus différents, le premier disponible dans les nœuds Nœud2 et Noeud4, le second disponible dans Nœud3

Enfin pour stocker les chemins déjà calculés par le contrôleur, une dernière table lie pour chaque couple d'identifiants de nœuds, représentant les extrémités d'un chemin, la liste des nœuds présents sur ce chemin. La Table 4.4 représente un exemple issue du Contrôleur1 de la Figure 4.1.

<Source, Destination>	**Chemin**
<Nœud2, Nœud4>	<Nœud2, Nœud4>
<Nœud1, Nœud3>	<Nœud1, Nœud2, Nœud3>

Table 4.4 – Exemple de table de chemins présente dans les contrôleurs SDN. Ici le Contrôleur1 de la Figure 4.1 possède les chemins du Nœud2 au Nœud4, ainsi que du Nœud1 au Nœud3

Messages de contrôle

Dans SRSC, les nœuds et le contrôleur ont besoin d'un canal de communication pour échanger des informations à propos de la topologie du réseau, des contenus disponibles, du voisinage, ainsi que des règles de routage. Ce canal a été établi en utilisant uniquement des messages *Interest* et *Data* définis dans NDN afin d'intégrer le plan de contrôle le plus naturellement possible.

Nous avons donc défini des préfixes qui sont réservés au plan de contrôle, ceux-ci sont présentés dans la Table 4.5. Ces messages sont identifiés par des messages NDN utilisant des préfixes pré-définis, c'est à dire des noms spécifiques.

Messages	Préfixes	Étapes
InterestBoot	`/allNodes/controller/<IdController>`	controller announce
InterestDiscover	`/neighbours/node/<IdNode>`	neighbours discovery
InterestNeighbours	`/controller/<IdController>/node/<IdNode>/<Neighbours>`	topology discovery
InterestNewContent	`/controller/<IdController>/incache/node/<IdNode>/data/<Data>`	content announcement
InterestPath	`/controller/<IdController>/path/node/<IdNode>/data/<Data>`	rules asking
InterestCreatePath	`/node/<IdNextNode1>/install/<IdNextNode2>/.../<IdNode2>/data/<Data>`	path establishing
InterestDeleteContent	`/controller/<IdController>/deletecache/node/<IdNode>/data/<Data>`	removal of content

Table 4.5 – *Interest* de Contrôle de SRSC défini pour le canal de communication entre les nœuds et les contrôleurs. Les *Data* de contrôle ont les mêmes préfixes et sont utilisés pour transporter des informations si nécessaire.

Deux types de messages de contrôle sont identifiés : les messages de requête et les annonces. Les messages de requête sont des *Interest* de contrôle qui nécessitent un message *Data* de contrôle en retour pour transporter des informations complémentaires, comme par exemple les messages *InterestDiscover* (demandant l'identité des voisins) et *InterestPath* (demandant au contrôleur le chemin pour un contenu), auxquels seront retournés des messages *DataDiscover* (contenant l'identité du nœud répondant) et *DataPath* (contenant le chemin pour accéder au contenu). Les messages d'annonce sont des *Interest* de contrôle dans lesquels les informations sont directement insérées dans le préfixe (*piggybacking*) et aucun message *Data* n'est alors retourné, réduisant le nombre de messages émis.

Les préfixes de ces messages de contrôle utilisent des champs pour transporter des informations comme par exemple <IdController> et <IdNode> qui permettent de renseigner l'identifiant du contrôleur et du nœud. Le champ <Neighbours> correspond à une liste de couples (<IdNode>\<IdController>) afin de connaître les voisins d'un nœud et déterminer les différents domaines du réseau et donc les nœuds de bordure. <Data> fait référence au préfixe du contenu demandé par les utilisateurs pour lequel, par exemple, devront être installées des règles dans la FIB d'un nœud.

Lorsqu'un *InterestDiscover* est envoyé, un message *DataDiscover* avec le même préfixe est retourné, transportant un couple (<IdNode>\<IdController>) permettant d'identifier les nœuds voisins et le domaine auquel ils appartiennent. En retour d'un *InterestPath*, le message *DataPath* transporte un chemin entre la source et la destination, sous forme de liste d'identifiants de nœuds.

Les algorithmes utilisés par le contrôleur et les nœuds pour chacune des deux phases (*Bootstrapping* et *Rules Forwarding*) utilisent ces messages de contrôle pour transmettre les informations leur permettant de fonctionner et sont présentés dans les sections suivantes. Afin de mieux visualiser chaque étape de ces algorithmes, nous les illustrons par un exemple détaillé présentant le fonctionnement de SRSC avec les différents envois de messages de contrôle et leurs préfixes associés.

4.3.2 Phase d'amorçage (i.e. *Bootstrapping*) :

Lors de la phase d'amorçage, les nœuds vont se lier au contrôleur qui va pouvoir ainsi découvrir la topologie du réseau qu'il gère et récupérer les informations sur la localisation des contenus. Un réseau peut être administré par plusieurs contrôleurs, où chacun sera en charge d'un domaine, comme illustré sur la figure 4.1, pouvant être perçu comme un AS. Si deux contrôleurs sont déployés au sein d'un AS, ce dernier sera à nouveau découpé en deux domaines, ou sous-AS.

Au démarrage de cette phase, les contrôleurs envoient un message contenant leur identité pour s'annoncer au réseau. À la réception de ce message, les nœuds vont pouvoir connaître le contrôleur auquel ils se lient. Dans le cas où plusieurs contrôleurs sont déployés dans un réseau, le nœud se lie au contrôleur duquel il a reçu l'annonce en premier. Il pourrait y avoir des mécanismes plus sophistiqués de sélection du contrôleur que l'on garde pour des travaux futurs.

Par ailleurs, la réception de ce message permet à chaque nœud de connaître l'interface (face) avec laquelle communiquer avec son contrôleur et d'installer les règles dans les FIB pour transmettre les messages destinés au contrôleur. Une fois le premier message contenant l'identité d'un contrôleur reçu par un nœud, ce dernier ignore les annonces potentielles reçues d'autres contrôleurs.

Chaque nœud démarre ensuite une phase de découverte de son voisinage et envoie ces informations à son contrôleur (identités de ses voisins ainsi que leurs contrôleurs associés). Celui-ci va alors pouvoir construire la topologie du domaine dont il a la charge.

Cette étape permet aussi de détecter les nœuds de bordure qui délimitent le domaine. Du point de vue du contrôleur, les nœuds de bordure sont ceux interconnectés avec un des nœuds appartenant à son domaine, mais qui sont gérés par un autre contrôleur, donc dans un autre domaine. Par exemple sur la Figure 4.1, le Nœud5 sera le nœud de bordure du Contrôleur1, et le Nœud4 celui du Contrôleur2. Ces nœuds de bordure servent de passerelles entre les différents domaines lorsqu'un contrôleur souhaite envoyer une requête dans un domaine adjacent.

Une fois la phase d'amorçage terminée, le canal de communication entre les nœuds et leur contrôleur est mis en place. Les nœuds vont donc pouvoir interroger le contrôleur pour demander les chemins vers les différents contenus.

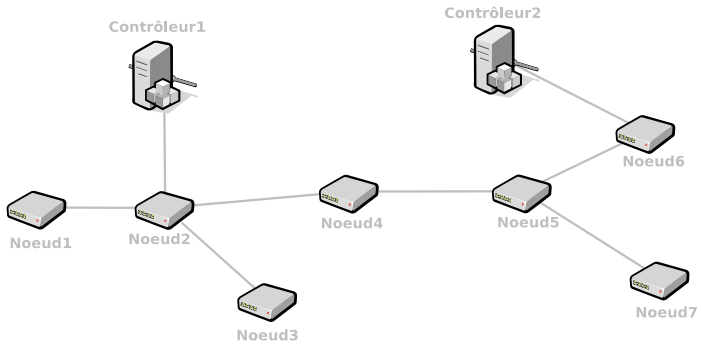


Figure 4.2 – Topologie du réseau utilisée dans cet exemple

Pour notre exemple, la topologie est composée de sept nœuds NDN et de deux contrôleurs comme représenté sur la Figure 4.2. Nous prenons deux contrôleurs pour illustrer tous les cas possibles du protocole et montrer l'intérêt des nœuds de bordure. Si un réseau est géré uniquement par un contrôleur, il n'y a, bien sûr, aucun nœud de bordure et certains passages des algorithmes ne sont plus nécessaires. Nous faisons l'hypothèse que chaque lien du réseau possède la même capacité afin de montrer, pas à pas, les liens parcourus par chaque message. La phase de *Bootstrapping* pour cet exemple se compose de huit étapes détaillées ci-après.

Algorithm 1 *Bootstrapping* de SRSC pour les Contrôleurs

//Start Bootstrapping
1. Controller.broadcast(InterestBoot)
2. **For All** InterestNeighbours received from Node Ni **do :**
3. Controller.Update(Topology)
4. **If** Neighbour.GetControllerID() != Controller **do :**
5. Controller.AddBorderNode(Neighbour)
6. **End If**
7. **End For**

//Content Tracking
For All InterestNewContent received from Node Ni **do :**
9. Controller.AddNewContent(NewContent, Ni)
10. **End For**

Algorithm 2 *Bootstrapping* de SRSC pour les nœuds

//Topology discovery
1. Node receives InterestBoot from Controller
2. **If** Node.AlreadyBindToController() **do :**
3. Node.drop(InterestBoot)
4. **Else**
5. Node.BindTo(Controller)
6. Node.broadcast(InterestBoot)
7. Node.Broadcast(InterestDiscover)
8. **While** Node.NbrDataDiscoverReceived() !=Node.NbrFaces() **do :**
9. Node receives DataDiscover from Neighbour
10. Node.AddInNeighboursList(Neighbour)
11. **End While**
12. Node.Send(Controller,InterestNeighbours(NeighboursList))
13. **End If**
14. **For All** InterestDiscover received from Node Ni **do :**
15. Node.Send(Ni, DataDiscover(Node,Controller))
16. **End For**

//Content Location
17. **For All** NewContent available **do :**
18. Node.Send(Controller,InterestNewContent(Node,NewContent))
19. **End For**

Étape 1

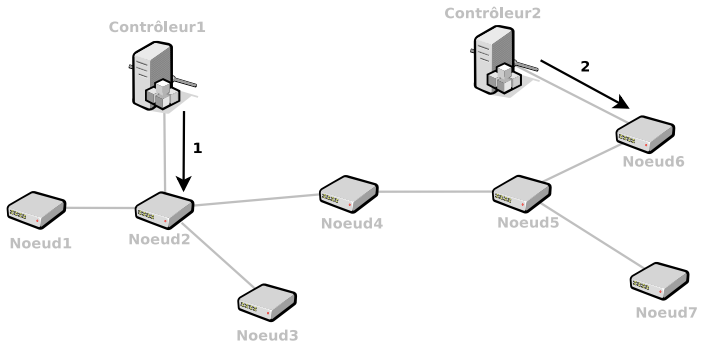


Figure 4.3 – *Bootstrapping* : Étape 1

Le protocole SRSC débute la phase de *Bootstrapping* par l'envoi d'un *InterestBoot* par chaque contrôleur (Algorithme 1 ligne 1). Cet *InterestBoot* contient l'identité du contrôleur. Dans notre exemple (Figure 4.3), les identifiants des nœuds et contrôleurs seront leurs noms, comme indiqué sur le schéma Figure 4.3 (Contrôleur1, Contrôleur2, Nœud1, ...). Les *InterestBoot* envoyés sont donc les suivants :

— 1 : *InterestBoot* `/allNodes/controller/Contrôleur1`
— 2 : *InterestBoot* `/allNodes/controller/Contrôleur2`

Étape 2

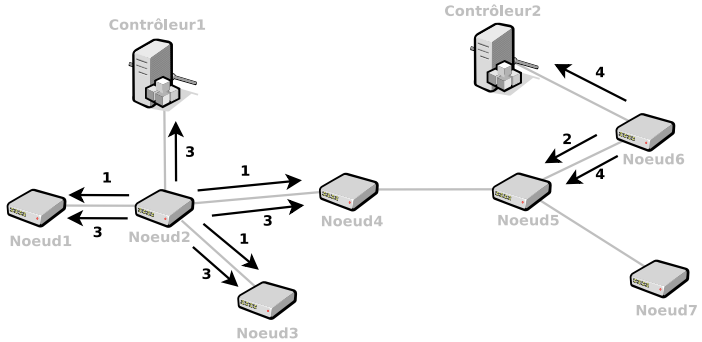


Figure 4.4 – *Bootstrapping* : Étape 2

Lorsqu'un nœud reçoit un *InterestBoot* (Figure 4.4), s'il est déjà lié à un contrôleur, il ne tient pas compte de ce message (Algorithme 2 ligne 2). Sinon, le nœud se lie au contrôleur (Algorithme 2 ligne 5) et émet l'*InterestBoot* sur tous les autres liens (Algorithme 2 ligne 6).

Ici, le Nœud2 sera alors lié au Contrôleur1 et le Nœud6 au Contrôleur2. Ils vont ensuite propager cet *InterestBoot* dans le réseau (messages 1 et 2).

— 1 : *InterestBoot* `/allNodes/controller/Contrôleur1`

— 2 : *InterestBoot* `/allNodes/controller/Contrôleur2`

Les nœuds Nœud2 et Nœud6 lancent alors le processus de découverte des voisins (Algorithme 2 ligne 7) en diffusant des *InterestDiscover* (messages 3 et 4).

— 3 : *InterestDiscover* `/neighbours/node/Noeud2`

— 4 : *InterestDiscover* `/neighbours/node/Noeud6`

Étape 3

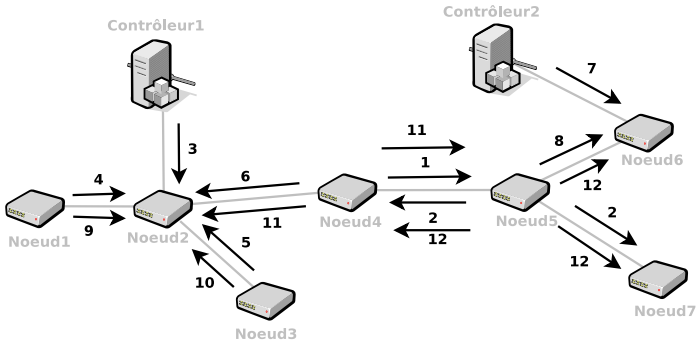


Figure 4.5 – *Bootstrapping* : Étape 3

Les nœuds Nœud1, Nœud3 et Nœud4 vont se lier au Contrôleur1 et le Nœud4 continue de transmettre l'*InterestBoot* du Contrôleur1 (message 1) (Figure 4.5).

— 1 : *InterestBoot* `/allNodes/controller/Contrôleur1`

Le Nœud5 se lie au Contrôleur2 et émet l'*InterestBoot* sur ces 2 interfaces (messages 2).

— 2 : *InterestBoot* `/allNodes/controller/Contrôleur2`

Lorsqu'un nœud reçoit un *InterestDiscover*, il répond avec un message *DataDiscover* contenant son identité et celle de son contrôleur (Algorithme 2 ligne 15), comme c'est le cas ici pour les nœuds Nœud1, Nœud3, Nœud4, Nœud5, ainsi que les deux contrôleurs (messages 3 à 8).

— 3 : *DataDiscover* `/neighbours/node/Noeud2`, donnée : **Contrôleur1/Contrôleur1**

— 4 : *DataDiscover* `/neighbours/node/Noeud2`, donnée : **Noeud1/Contrôleur1**

— 5 : *DataDiscover* `/neighbours/node/Noeud2`, donnée : **Noeud3/Contrôleur1**

— 6 : *DataDiscover* `/neighbours/node/Noeud2`, donnée : **Noeud4/Contrôleur1**

— 7 : *DataDiscover* `/neighbours/node/Noeud6`, donnée : **Contrôleur2/Contrôleur2**

— 8 : *DataDiscover* `/neighbours/node/Noeud6`, donnée : **Noeud5/Contrôleur2**

Les nœuds Nœud1, Nœud3, Nœud4 et Nœud 5 démarrent le processus de découverte des voisins et émettent chacun un *InterestDiscover* (messages 9 à 12).

— 9 : *InterestDiscover* `/neighbours/node/Noeud1`
— 10 : *InterestDiscover* `/neighbours/node/Noeud3`
— 11 : *InterestDiscover* `/neighbours/node/Noeud4`
— 12 : *InterestDiscover* `/neighbours/node/Noeud5`

Étape 4

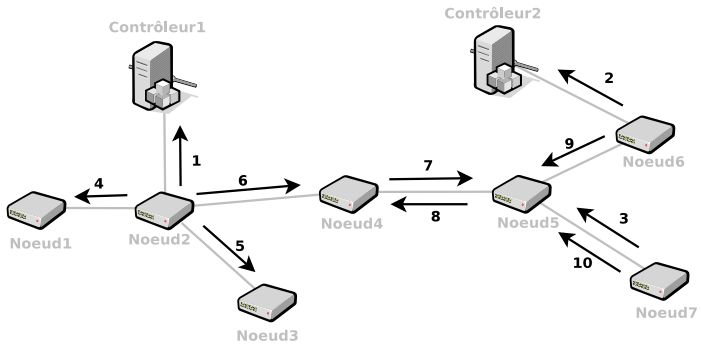


Figure 4.6 – *Bootstrapping* : Étape 4

Le Nœud4 reçoit l'*InterestBoot* du Nœud5 et le Nœud5 celui du Nœud4, mais étant déjà tous les deux liés à des contrôleurs, ils suppriment ces *Interest* (Algorithme 2 ligne 2) (Figure 4.6).

Les nœuds ayant émis un *InterestDiscover* attendent ensuite que chacun de leurs voisins répondent avec un message *Data* (Algorithme 2 ligne 9) pour récupérer les identités de leurs voisins ainsi que celles de leurs contrôleurs associés (Algorithme 2 ligne 10).

Une fois la réponse de tous ses voisins reçue, le nœud envoie un *InterestNeighbours* (contenant la liste des voisins et l'identité de leur contrôleur) à son contrôleur pour l'avertir de sa vision locale du réseau (Algorithme 2 ligne 12). Ici les nœuds Nœud2 et Nœud6 connaissent leurs voisins et peuvent donc envoyer à leur contrôleur un *InterestNeighbours* (messages 1 et 2).

— 1 : *InterestNeighbours* `/controller/Contrôleur1/node/Noeud2/Contrôleur1\Contrôleur1/Noeud1\Contrôleur1/Noeud3\Contrôleur1/Noeud4\Contrôleur1`
— 2 : *InterestNeighbours* `/controller/Contrôleur2/node/Noeud6/Contrôleur2\Contrôleur2/Noeud5\Contrôleur2`

Le Nœud7 lance la découverte de ses voisins en envoyant un *InterestDiscover* (message 3).

— 3 : *InterestDiscover* `/neighbours/node/Noeud7`

Les nœuds Nœud2, Nœud4, Nœud5, Nœud6 et Nœud7, quant à eux, répondent à la réception des messages *InterestDiscover* de leurs voisins par un *DataDiscover* (messages 4 à 10).

— 4 : *DataDiscover* `/neighbours/node/Noeud1`, donnée : **Noeud2/Contrôleur1**

— 5 : *DataDiscover* `/neighbours/node/Noeud3`, donnée : **Noeud2/Contrôleur1**
— 6 : *DataDiscover* `/neighbours/node/Noeud4`, donnée : **Noeud2/Contrôleur1**
— 7 : *DataDiscover* `/neighbours/node/Noeud5`, donnée : **Noeud4/Contrôleur1**
— 8 : *DataDiscover* `/neighbours/node/Noeud4`, donnée : **Noeud5/Contrôleur2**
— 9 : *DataDiscover* `/neighbours/node/Noeud5`, donnée : **Noeud6/Contrôleur2**
— 10 : *DataDiscover* `/neighbours/node/Noeud5`, donnée : **Noeud7/Contrôleur2**

Étape 5

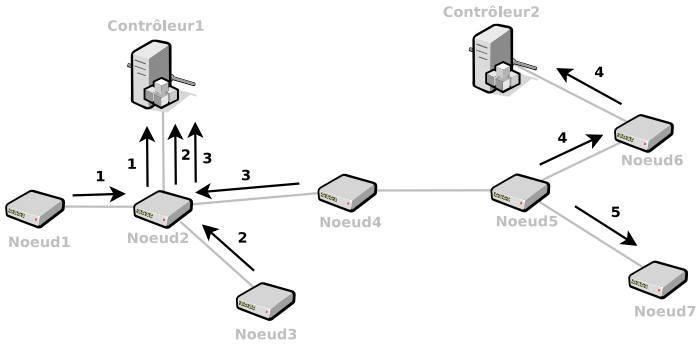


Figure 4.7 – *Bootstrapping* : Étape 5

Lorsqu'un contrôleur reçoit un *InterestNeighbours* d'un nœud (Figure 4.7), il ajoute, si besoin, les liens décrits dans la liste représentée par le champ <Neighbours> à sa topologie (Algorithme 1 ligne 3). Ensuite, s'il remarque qu'un nœud est géré par un autre contrôleur, ce nœud est ajouté comme nœud de bordure (Algorithme 1 ligne 5).

Ici, le Contrôleur1 voit que le Nœud2 et ses voisins sont dans son domaine, il n'y aura pour le moment pas de nœud de bordure. Même chose pour le Contrôleur2 avec le Nœud6. Les nœuds Nœud1, Nœud3, Nœud4 et Nœud5 reçoivent maintenant la réponse de leurs voisins à l'*InterestDiscover* et peuvent émettre à leur tour leur vision locale vers leurs contrôleurs grâce à un *InterestNeighbours* (messages 1 à 4).

— 1 : *InterestNeighbours* `/controller/Contrôleur1/node/Noeud1/Noeud2\Contrôleur1`
— 2 : *InterestNeighbours* `/controller/Contrôleur1/node/Noeud3/Noeud2\Contrôleur1`
— 3 : *InterestNeighbours*
`/controller/Contrôleur1/node/Noeud4/Noeud2\Contrôleur1/Noeud5\Contrôleur2`
— 4 : *InterestNeighbours*
`/controller/Contrôleur2/node/Noeud5/Noeud6\Contrôleur2/Noeud7\Contrôleur2/Noeud4\Contrôleur1`

Le Nœud5 répond à l'*InterestDiscover* du Nœud7 (message 5) en lui envoyant sont identité et celle de son contrôleur dans un message *DataDiscover*.

— 5 : *DataDiscover* `/neighbours/node/Noeud7`, donnée : **Noeud5/Contrôleur2**

Étape 6

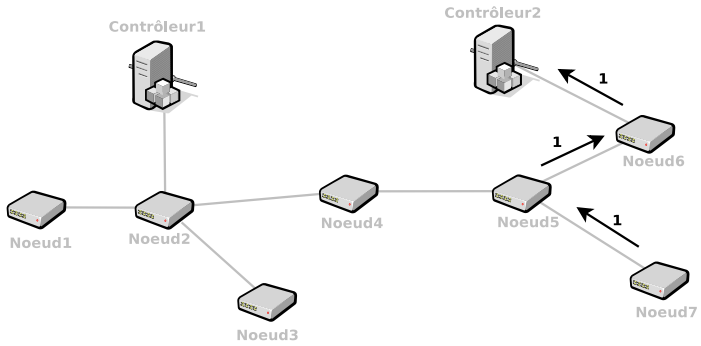


Figure 4.8 – *Bootstrapping* : Étape 6

A la réception de l'*InterestNeighbours* du Nœud4, le Contrôleur1 voit que le Nœud5 est lié au Contrôleur2, il sera donc un nœud de bordure et servira de passerelle pour contacter le réseau géré par le Contrôleur2. De son coté, le Contrôleur2 apprend que le Nœud4 est lié au Contrôleur1, il deviendra donc son nœud de bordure pour le réseau du Contrôleur1. Finalement, le Nœud7 envoie, lui aussi, sa vision locale au contrôleur avec un *InterestNeighbours* (message 1) (Figure 4.8).

— 1 : *InterestNeighbours* `/controller/Contrôleur2/node/Noeud7/Noeud5\Contrôleur2`

Étape 7

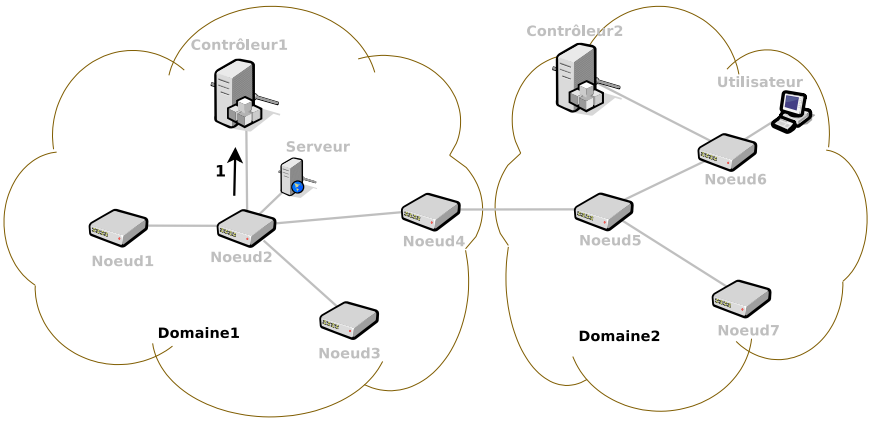


Figure 4.9 – *Bootstrapping* : Étape 7

Le réseau est maintenant séparé en plusieurs domaines, chacun géré par un contrôleur comme cela est représenté sur la Figure 4.9. Pour finir cette phase de *Boostrapping*, pour chaque contenu

qu'un nœud a à disposition, il avertit sont contrôleur grâce à un *InterestNewContent* dans le but de localiser le contenu (Algorithme 2 ligne 18).

Dans notre exemple, un serveur mettant à disposition le contenu ayant pour préfixe `/serveur/video/exemple/chien.avi` est placé sur le Nœud2. Ce dernier va alors prévenir son contrôleur en émettant le message *InterestNewContent* ayant le préfixe suivant :

— 1 : *InterestNewContent*
`/controller/Contrôleur1/incache/node/Noeud2/data/serveur/video/exemple/chien.avi`

Pour chaque *InterestNewContent* reçu, le contrôleur ajoute dans sa table de contenus une entrée pour le contenu annoncé et le nœud ayant émis le message (Algorithme 1 ligne 9).

Évaluation du coût en signalisation (overhead)

Nous allons maintenant évaluer le coût en signalisation, c'est à dire en matière de nombre de messages émis par ces deux algorithmes, lors de la phase de *Boostrapping* pour un réseau NDN constitué de n nœuds et d'un contrôleur gérant le réseau. Nous réalisons cette estimation dans un réseau géré par un unique contrôleur car aucun échange n'ayant lieu entre les différents domaines, ce coût dépendra uniquement du nombre de nœuds gérés par chaque contrôleur.

Afin de connaître le coût en signalisation maximal de ces algorithmes, nous considérons un réseau totalement maillé, où chaque nœud est directement connecté à l'ensemble des autres nœuds du réseau (i.e. chaque nœud est atteignable en un saut). L'annonce du contrôleur va donc générer $n * n$ messages sur les liens (envoi par le contrôleur aux n nœuds puis retransmission par chacun des nœuds sur l'ensemble des liens).

Ensuite, lors de la découverte des voisins, à nouveau $n*n$ messages seront émis pour découvrir les voisinages (chaque nœud émettant un message vers l'ensemble des autres nœuds ainsi que vers le contrôleur). Les réponses de cette phase de découverte entraîneront $n * n$ messages une fois de plus. Enfin, afin d'avertir le contrôleur de la topologie qu'il gère, chaque nœud va envoyer un message, générant n messages.

Donc au total, dans une topologie complètement connectée, cette phase de *Bootstrapping* représente un volume de messages émis maximal égal à :

$$3n^2 + n \tag{4.1}$$

Si nous effectuons maintenant cette même estimation dans un réseau linéaire (un seul chemin possible pour chaque destination), le message d'annonce du contrôleur va générer n messages. La découverte des voisins va générer $2 * n - 1$ messages.

Les réponses à cette phase vont générer $2 * n - 1$ messages en retour. Et enfin, pour avertir le contrôleur, les n nœuds vont émettre un message au contrôleur, générant $\sum_{i=0}^{n} i$. Dans cette topologie linéaire, la phase de *Bootstrapping* représente un volume de messages émis maximal de :

$$n + 2 * (2 * n - 1) + \sum_{i=0}^{n} i \tag{4.2}$$

Soit :

$$(n * (n + 1))/2 - 2 \tag{4.3}$$

Par exemple, dans une topologie de 20 nœuds connectés, nous aurons donc 1 220 messages échangés lors de la phase de *Bootstrapping*. Si la topologie est linéaire, nous aurons 308 messages échangés. Ces deux topologies représentent les deux cas extrêmes possibles et le coût en nombre de messages reste limité pour une phase nécessaire seulement lors de la mise en route dans le réseau, et qui ne sera pas rejouée périodiquement.

4.3.3 Phase d'acheminement des règles (i.e. *Rules Forwarding*) :

La phase de *Rules Forwarding* consiste à remplir au besoin les FIB des nœuds avec des règles de *forwarding*. Ces règles vont ensuite permettre de transmettre les *Interest* vers la destination la plus proche possédant le contenu demandé (serveur d'origine du contenu ou *Content Store* d'un nœud).

Lorsqu'un nœud doit transmettre un *Interest* et qu'il ne possède pas de règle pour celui ci, il interroge son contrôleur. Le contrôleur calcule alors le chemin le plus court entre le nœud source et la destination possédant le contenu le plus proche grâce à l'algorithme de Dijkstra. Il envoie ensuite le chemin calculé au nœud. Le nœud source va pouvoir installer les règles dans sa FIB ainsi que dans celle des nœuds du chemin en faisant transiter un message de contrôle *InterestCreatePath* vers la destination, puis il envoie l'*Interest* qui sera acheminé vers cette dernière. Ainsi, tous les nœuds du chemin n'auront pas à contacter à leur tour le contrôleur mais vont installer la règle dans leur FIB à la réception du message *InterestCreatePath*, diminuant le nombre de messages dans le réseau.

Le contrôleur a donc connaissance de tous les contenus disponibles dans son domaine et si un contenu demandé n'y apparaît pas, le contrôleur va alors rediriger le message*Interest* vers les nœuds de bordure, de sorte à envoyer la requête dans d'autres domaines du réseau pour tenter d'atteindre le contenu. Une fois le message *Interest* relayé dans un autre domaine, un autre contrôleur sera en charge de diriger cette requête au sein de son réseau, où vers un autre nœud de bordure, de proche en proche, jusqu'à atteindre la donnée.

Algorithm 4 *Rules Forwarding* de SRSC pour les contrôleurs

```
1. For InterestPath received from Node Ni do :
2.  RequestedContent ← ExtractPrefixe(InterestPath)
3.  If ContentList.Has(RequestedContent) do :
4.   Destination ← GetNode(RequestedContent)
5.  Else
6.   Destination ← GetBorderNode()
7.  End If
8.   Path ← Controller.GetPath(Ni, Destination)
9.   Controller.Send(Ni, Path)
10. End For
```

Cette phase de *Rules Forwarding* représente la partie cruciale de SRSC pendant laquelle les nœuds vont pouvoir demander des règles de *forwarding* de manière réactive au contrôleur pour acheminer les requêtes vers les contenus les plus proches. Elle se décompose en quatre étapes dans l'exemple présenté. L'Algorithme 3 correspond à la phase de *Rules Forwarding* coté nœud et l'Algorithme 4 coté contrôleur.

Algorithm 3 *Rules Forwarding* de SRSC pour les nœuds

```
//Send Request to Controller
1. For All Interest(Content) received do :
2.  If Node.IsInCS(Content) then
3.    Node.Send(Content)
4.  Else If Node.AlreadyInPIT(Content) do :
5.      Node.UpdateEntry(PIT,Content)
6.    Else If Node.AlreadyInFIB(Content) do :
7.       Node.Send(FIB,Content)
8.       Node.AddEntry(PIT,Content)
9.      Else
10.          Node.Send(Controller,InterestPath(Node,Content))
11.      End If
12.    End If
13.  End If
14. End For

//Receive Path from Controller
15. For All DataPath received from Controller do :
16.  NextNode ← Node.ExtractNextNode(DataPath)
17.  Content ← NodeExtractContent(DataPath)
18.  Node.AddEntry(FIB,Content,NextNode)
19.  Node.Send(NextNode,InterestCreatePath)
20.  Node.Send(NextNode,Content)
21. End For
22. For All InterestCreatePath received from Ni do :
23.  NextNode ← Node.ExtractNextNode(InterestCreatePath)
24.  Content ← NodeExtractContent(InterestCreatePath)
25.  Node.AddEntry(FIB,Content, NextNode)
26.  If not Node.IsLastNode(InterestCreatePath) do :
27.    Node.Send(NextNode,InterestCreatePath)
28.  End If
29. End For
```

Étape 1

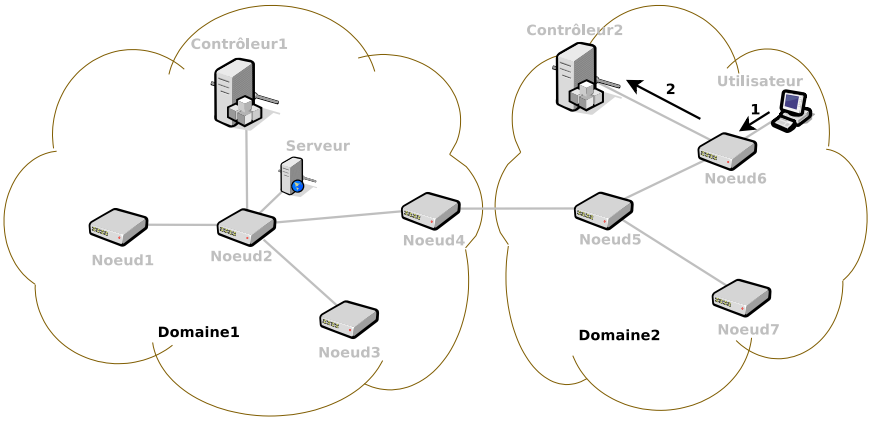


Figure 4.10 – *Rules Forwarding* : Étape 1

Une fois la phase de *bootstrapping* terminée, les contrôleurs connaissent les nœuds qu'ils doivent gérer, ainsi que leurs nœuds de bordure pour atteindre les autres domaines comme indiqué sur la Figure 4.10.

Maintenant, pour détailler la phase de *Rules forwarding*, admettons qu'un utilisateur soit situé dans le Domaine2 du réseau, connecté au Nœud6. Cet utilisateur souhaite accéder au contenu ayant le nom `"/serveur/video/exemple/chien.avi"`, proposé par le serveur qui lui, est placé dans le Domaine1 du réseau. Pour accéder au contenu, l'utilisateur émet un message *Interest* avec le préfixe du contenu demandé (message 1).

— 1 : *Interest* `/serveur/video/exemple/chien.avi`

Lorsqu'un nœud reçoit un message *Interest*, il regarde si le contenu demandé est disponible dans son *Content Store* pour répondre directement à la requête (Algorithme 3 ligne 2) ou, s'il a déjà émis le même *Interest* et attend la réponse, met à jour sa PIT (Algorithme 3 ligne 5) de façon à transmettre la donnée vers l'ensemble des clients la demandant. Dans le cas contraire, il regarde dans sa FIB s'il possède une entrée pour émettre cet *Interest* (Algorithme 3 ligne 7-8) et ajouter une entrée dans sa PIT.

Finalement si aucune condition précédente n'est remplie, le nœud va contacter son contrôleur pour lui demander des règles de *forwarding*. Il lui envoie un *InterestPath* contenant le préfixe de la donnée pour laquelle il désire établir le chemin (Algorithme 3 ligne 10). Dans notre exemple, le Nœud6 reçoit le message *Interest* mais ne possède pas le contenu ni les règles pour y accéder, il va donc envoyer à son contrôleur un *InterestPath* (message 2).

— 2 : *InterestPath*
`/controller/Contrôleur2/path/node/Noeud6/data/serveur/video/exemple/chien.avi`

Étape 2

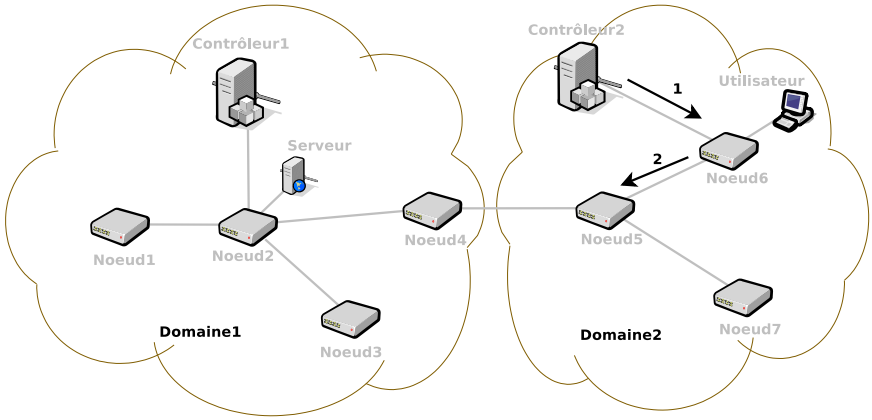


Figure 4.11 – *Rules Forwarding* : Étape 2

Lorsqu'un contrôleur reçoit un *InterestPath* d'un nœud, il extrait le préfixe du contenu pour lequel le nœud demande une règle de *forwarding* (Algorithme 4 ligne 2) et regarde dans sa table de contenus si, dans son domaine, un nœud possède ce contenu (Algorithme 4 ligne 4). Le contenu pouvant être répliqué dans plusieurs *Content Store*, le contrôleur choisit le plus court chemin (qui est aussi le moins coûteux dans notre cas).

Pour cela, un poids est défini sur chaque lien, permettant de les pondérer. Dans notre cas, chaque lien possède un poids de 1, revenant à choisir le plus court chemin en nombre de sauts, soit le nœud le plus proche de la source. Si le contenu n'est pas disponible, il envoie la requête vers un autre domaine en la transmettant à un nœud de bordure (Algorithme 4 ligne 6). Dans ce cas, un autre contrôleur sera en charge de trouver le contenu en suivant le même schéma. Le contrôleur calcule donc le plus court chemin entre la source (nœud qui a envoyé le message *InterestPath*) et la destination (nœud le plus proche qui possède le contenu ou un nœud de bordure) et envoie ce chemin dans un message *DataPath*. À la réception de l'*InterestPath* (Figure 4.11), le Contrôleur2 déduit que la donnée n'est pas disponible dans son domaine, il calcule donc le chemin entre le Nœud6 et le nœud de bordure (Nœud4). Il envoie les règles au Nœud6 en répondant à l'*InterestPath* avec un message *DataPath* (message 1).

— 1 : *DataPath*
`/controller/Contrôleur2/path/node/Noeud6/data/serveur/video/exemple/chien.avi`
donnée : **Noeud6_Noeud5_Noeud4**

Le Nœud6 installe les règles dans sa FIB en spécifiant que pour accéder au contenu ayant pour préfixe `"/serveur/video/exemple/chien.avi"`, il doit envoyer la requête au Nœud5. Il envoie ensuite un *InterestCreatePath* au Nœud5 (message 2) de sorte à installer les règles sur le chemin jusqu'au Nœud4. Le Nœud5 fait de même : il installe la règle pour transmettre la requête au Nœud4. Voyant que le Nœud4 est le prochain saut, il ne lui transfère pas l'*InterestCreatePath* puisque le Nœud4 est au bout du chemin.

— 2 : *InterestCreatePath*
`/node/Noeud5/install/Noeud5/Noeud4/data/serveur/video/exemple/chien.avi`

Comme précisé ci-dessus, il est possible d'attribuer des poids à chaque lien, de sorte à définir différentes politiques de *forwarding* en fonction de ces métriques, permettant de limiter les coûts pour les opérateurs ou effectuer de l'équilibrage de charge (*load balancing*).

Étape 3

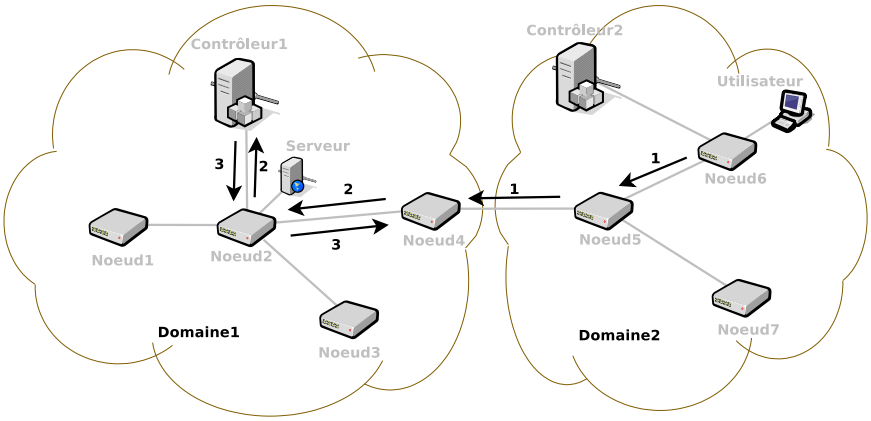


Figure 4.12 – *Rules Forwarding* : Étape 3

À la réception de la réponse du contrôleur, soit à la réception du *DataPah* (Algorithme 3 ligne 15), le nœud extrait le prochain saut pour atteindre le contenu (Algorithme 3 ligne 16), ainsi que le préfixe du contenu pour lequel installer la règle de *forwarding* (Algorithme 3 ligne 17). Il ajoute l'entrée dans sa FIB (Algorithme 3 ligne 18) et émet un *InterestCreatePath* à ce prochain saut pour installer les règles sur l'ensemble du chemin jusqu'à la destination (Algorithme 3 ligne 19). Il envoie ensuite l'*Interest* pour accéder au contenu sur le chemin précédemment créé (Algorithme 3 ligne 20).

Enfin, pour chaque *InterestCreatePath* reçu, le nœud extrait le prochain saut du préfixe, ainsi que le contenu pour lequel installer la règle de *forwarding*, et ajoute l'entrée dans sa FIB (Algorithme 3 ligne 23 à 25). S'il n'est pas le dernier nœud du chemin, il envoie l'*InterestCreatePath* au nœud suivant (Algorithme 3 ligne 27), sinon il n'en tient pas compte. Une fois les règles installées, le Nœud6 émet donc l'*Interest* pour le contenu `"/serveur/video/exemple/chien.avi"` (message 1) qui sera transmis jusqu'au Nœud4 (Figure 4.12).

— 1 : *Interest* `/serveur/video/exemple/chien.avi`

À son tour, le Nœud4 contacte son contrôleur avec un *InterestPath* (message 2) pour avoir les règles permettant d'acheminer la requête jusqu'au contenu.

— 2 : *InterestPath*
`/controller/Contrôleur1/path/node/Noeud4/data/serveur/video/exemple/chien.avi`

Le Contrôleur1 sait que le contenu est disponible dans le serveur au Nœud2 et répond donc à l'*InterestPath* avec un message *DataPath* (message 3) contenant le chemin entre le Nœud4 et le Nœud2.

— 3 : *DataPath*
`/controller/Contrôleur1/path/node/Noeud4/data/serveur/video/exemple/chien.avi`
donnée : **Noeud4_Noeud2**

Étape 4

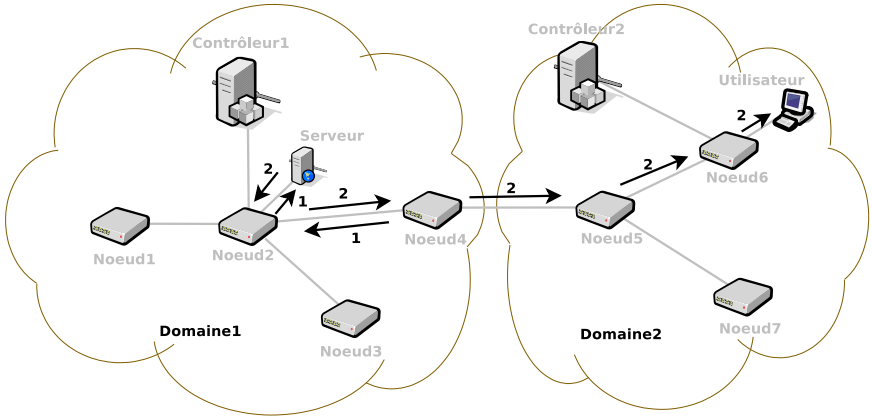


Figure 4.13 – *Rules Forwarding* : Étape 4

Après réception du chemin, le Nœud4 installe les règles dans sa FIB. Il sait que le Nœud2 possédant la donnée est son voisin grâce à sa liste de voisins donc il supprime l'*InterestCreatePath* et envoie l'*Interest* pour accéder au contenu au Nœud2 (message 1) (Figure 4.13).

— 1 : *Interest* `/serveur/video/exemple/chien.avi`

Celui ci récupère la donnée dans le serveur et retourne un message *Data* contenant la donnée demandée (message 2)

— 2 :*Data* `/serveur/video/exemple/chien.avi`
donnée : **la vidéo "chien.avi"**

Cette étape explique comment une donnée est transmise à un utilisateur grâce au contrôleur du réseau et sa connaissance omnisciente du réseau. Les nœuds du réseau peuvent bien entendu stocker les données qui transitent dans leur mémoire cache, et ce, en fonction des stratégies de mise en cache déployées dans les nœuds (typiquement LCD, LCE, LRU, etc.). Les nœuds du Domaine2 du réseau, vont donc potentiellement pourvoir garder une copie du contenu "`/serveur/video/exemple/chien.avi`" et avertir le Contrôleur2 que la donnée est maintenant disponible dans son domaine afin de rediriger les futures requêtes pour ce contenu vers ces nœuds. Cette redirection est rendue possible par les fonctionnalités de cache de NDN associées aux annonces périodiques entre nœuds et contrôleur lorsqu'une nouvelle donnée est stockée. Les contenus sont donc propagés dans le réseau, limitant le trafic et réduisant les délais des futurs utilisateurs pour y accéder.

Évaluation du coût en signalisation (*overhead*)

Afin d'évaluer la charge maximale théorique de messages que le contrôleur va recevoir, considérons à nouveau le réseau NDN entièrement maillé de n nœuds et d'un contrôleur utilisé précédemment pour évaluer la phase de *Boostrapping*.

En considérant que chaque nœud i reçoit r requêtes par secondes. Soit c le nombre de requêtes trouvant directement le contenu demandé dans le *Content Store* des nœuds et f le nombre de requêtes pour lesquelles une entrée dans la FIB est disponible pour transmettre la requête. On appelle h le nombre de contenus mis en cache par le nœud, générant un envoi d'avertissement au contrôleur. la charge du contrôleur en nombre de messages reçu par seconde est donc égale à :

$$\sum_{i=0}^{n} r_i - (c_i + f_i) + h_i \tag{4.4}$$

Ainsi, on peut remarquer que f va augmenter en fonction du temps, diminuant la charge du serveur. Ensuite, en maximisant c, et en minimisant h, la charge pourra être grandement diminuée, permettant au contrôleur d'être plus performant.

Ces deux phases permettent aux nœuds de se lier au contrôleur qui les administre et de prendre connaissance de la topologie du réseau. Le contrôleur peut ensuite appliquer les règles de transmission aux nœuds et permettre d'émettre les requêtes vers la copie de la donnée la plus proche de l'utilisateur. Il serait également possible de mettre en place d'autres politiques de routage avec d'autres métriques que la distance, ou permettant d'autres fonctionnalités comme l'équilibrage de charge, ou bien ignorer un nœud ou un contenu pour des raisons commerciales ou politiques , ou pour des raisons de sécurité.

4.4 Résumé

Dans ce chapitre, nous avons présenté notre protocole de routage SRSC (*SDN-based Routing Scheme for CCN/NDN*, adapté à l'architecture NDN. Notre protocole repose sur le paradigme SDN et utilise un contrôleur pour offrir un plan de contrôle à NDN et lui permettre de transmettre les requêtes vers les contenus les plus proches (*anycast*) et ce, sans avoir à inonder le réseau. SRSC utilise les messages *Interest* et *Data* de l'architecture NDN pour lesquels il réserve certains préfixes, afin de permettre aux nœuds et contrôleurs de communiquer. Le protocole SRSC s'affranchit ainsi totalement de l'environnement IP.

Nous avons présenté les grands principes de fonctionnement de notre protocole, ses différentes phases d'amorçage et de transmission des règles, ses algorithmes, formats des messages, et son coût en signalisation (*overhead*).

Nous évaluons les performances de notre protocole SRSC dans les deux chapitres suivants. Le chapitre 5 évalue SRSC via des simulations avec le logiciel NS-3. Le chapitre 6 décrit l'implantation de notre protocole SRSC dans notre plateforme expérimentale, ainsi que son évaluation et comparaison avec le protocole NLSR.

Chapitre 5

Expériences de simulation

Sommaire

5.1 Introduction 83
5.2 Environnement de simulation 83
5.2.1 Paramètres 84
5.2.2 Scénarios 86
5.2.3 Métriques 87
5.3 Résultats de l'évaluation 88
5.4 Résumé 94

5.1 Introduction

Dans le chapitre précédent, nous avons présenté notre protocole de routage SRSC pour l'architecture NDN basé sur le paradigme SDN. Nous avons détaillé son mode de fonctionnement ainsi que les algorithmes, et le format de ses messages.

Il convient dans ce chapitre d'évaluer les performances de notre protocole. Pour cela, nous avons choisi de l'évaluer à travers des expériences de simulation avec le logiciel à événements discrets NS-3 qui est largement utilisé dans la communauté de recherche. Nous avons également ajouté le module ndnSIM afin de pouvoir simuler l'architecture NDN [96, 97] et le comportement des nœuds. Nous décrivons par la suite l'environnement de simulation qui comprend les différents paramètres de simulation, scénarios et métriques d'évaluation.

5.2 Environnement de simulation

Afin d'évaluer notre protocole SRSC et sa capacité à envoyer les requêtes vers les mémoires caches les plus proches qui possèdent le contenu demandé, nous l'avons comparé avec la technique par défaut utilisée dans NDN, qui repose sur l'inondation du réseau ("*flooding*"). Cette méthode consiste à inonder le réseau des requêtes *Interest* en les diffusant sur toutes les interfaces des nœuds afin d'atteindre le contenu désiré.

5.2.1 Paramètres

Pour nos expériences de simulation, nous utilisons le simulateur à événements discrets NS-3 avec le module ndnSIM. Les paramètres de configuration sont ceux fréquemment retenus dans les travaux connexes sur l'évaluation de NDN via NS-3 [98, 52] et sont décrits ci-dessous puis récapitulés dans la Table 5.1.

Topologie

Pour modéliser la topologie du réseau NDN, nous avons choisi des topologies d'opérateurs existantes et disponibles librement : (i) la topologie Abilene qui comprend 11 nœuds ; (ii) la topologie Geant qui comprend 41 nœuds. Ces deux topologies sont présentées sur les figures 5.1 et 5.2.

Catalogue

Les différents contenus disponibles dans le réseau sont modélisés par un catalogue de 100 000 contenus. Les tailles des *Content Stores* peuvent stocker 100 contenus, soit 0.1% du catalogue.

Gestion des caches

Afin de gérer les contenus à conserver en cache, les nœuds reposent sur une stratégie de cache et une politique de remplacement des caches. La stratégie de cache consiste à décider si un contenu doit être conservé en mémoire ou non ; la politique de remplacement consiste à décider quel élément retirer du *Content Store* si de l'espace doit être libéré pour stocker un nouveau contenu. Notre stratégie de cache est LCD, *Leave-Copy Down* (LCD), celle par défaut dans NDN où chaque nœud stocke toujours tous les contenus reçus. Ainsi, chaque message *Data* est conservé en mémoire pour une réutilisation future. Notre politique de remplacement est LFU (*Least Frequently Used*), celle aussi par défaut dans NDN, où l'entrée la moins fréquemment utilisée dans le *Content Store* est supprimée.

Nombre d'utilisateurs et trafic

Nous déployons de manière uniforme 40 utilisateurs sur chacune des topologies Abilene et Geant. Chaque utilisateur génère 5 requêtes par seconde (*Interest*) vers des contenus. Le choix de ces contenus est déterminé par leur popularité ; cette popularité est modélisée via une loi de probabilité ZipF couramment utilisée dans la littérature, avec un paramètre $\alpha = 1.1$. Cette distribution et ce paramètre alpha sont représentatifs de la popularité des contenus généralement observés sur l'Internet et se retrouvent fréquemment dans d'autres études similaires utilisant NDN dans le simulateur NS-3/ndnSIM.

Chaque expérience de simulation sera effectuée 20 fois et les résultats présentés seront une moyenne sur laquelle nous présenterons les écarts types.

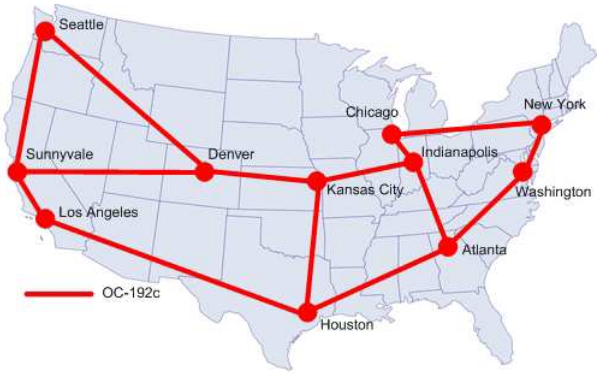


Figure 5.1 – Représentation de la topologie Abilene déployée au États-Unis et comprenant 11 nœuds [99]

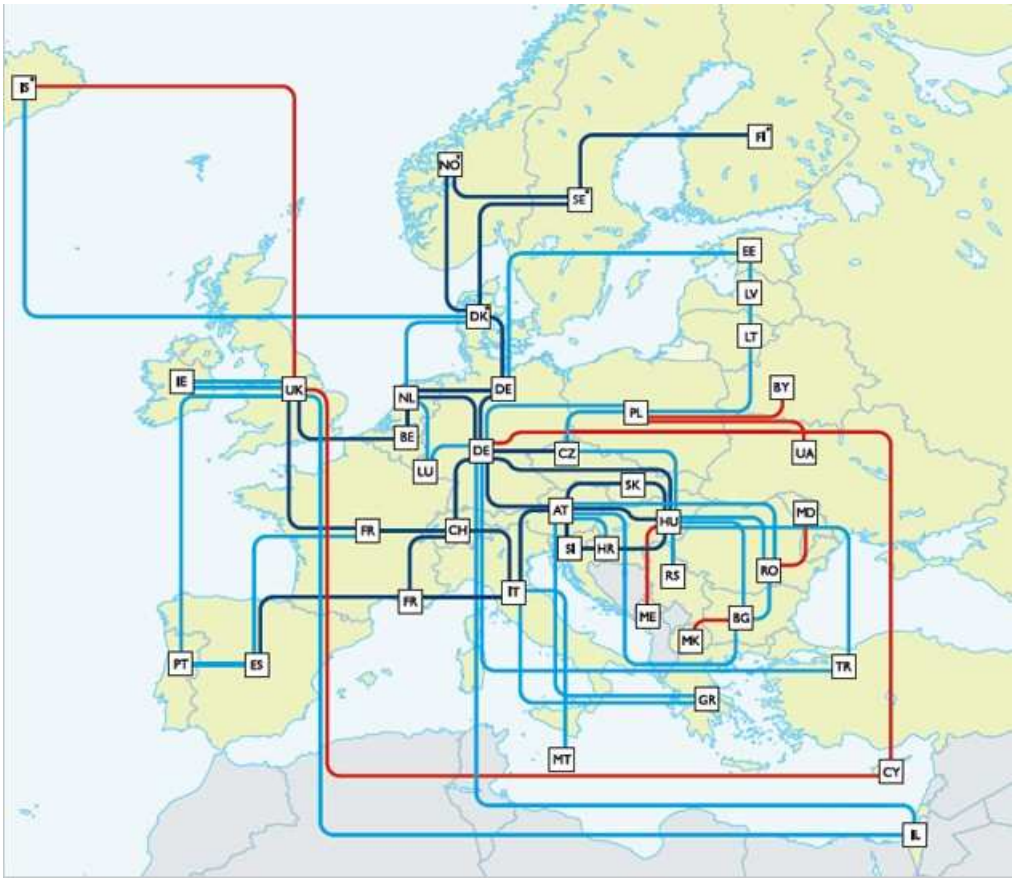


Figure 5.2 – Représentation de la topologie Geant déployée en Europe et comprenant 41 nœuds [100]

Paramètres	**Valeurs**
Topologies	ISP (Abilene, Geant)
Taille du catalogue	100.000 contenus
Nombre de catalogue	1
Nombre d'utilisateurs	40
Taille des caches	100 contenus (0,1% du catalogue)
Modèle de popularité	Zipf(α=1.1)
Fréquence des requêtes	5 par sec.
Stratégie de mise en cache	LCD
Politique de remplacement des caches	LFU

Table 5.1 – Paramètres de simulation

5.2.2 Scénarios

Au delà du nombre d'utilisateurs et de requêtes pour simuler les échanges dans un réseau, nous avons également évalué les performances de SRSC en fonction d'autres scénarios. En effet, NDN est un réseau de caches et il convient d'étudier l'impact des mémoires caches dans le réseau. Nous proposons ainsi plusieurs scénarios dans lesquels tous les nœuds ne sont pas nécessairement équipés de *Content Stores*, comme cela est représenté sur la Figure 5.3. Nous varions ainsi le nombre de nœuds équipés de *Content Stores* et ce nombre va de 0% (aucune mémoire de stockage) à 100% (tous les nœuds sont équipés de capacité de stockage). Nous distinguerons les scénarios avec une proportion de nœuds équipés de mémoire avec 0%, 20%, 40%, 50%, 80% et 100%.

Ainsi, dans le *scénario 0%*, il n'y a aucun *Content Store* dans le réseau. Les contenus ne peuvent donc pas être stockés par les nœuds du réseau et les messages *Interest* sont transmis jusqu'au serveur d'origine pour obtenir la donnée désirée. Ce scénario rappelle l'Internet dans lequel chaque requête doit être acheminée vers le serveur d'origine. Ce scénario ne pourra bénéficier des capacités de cache de l'architecture NDN, et servira de limite basse pour évaluer l'impact du nombre de *Content Stores* (pire cas). En effet il correspond au cas où aucune requête ne trouve son contenu dans une mémoire cache.

Au contraire, dans le *scénario 100%*, tous les nœuds possèdent un *Content Store*, ce qui correspond au scénario classique de NDN dans lequel tous les nœuds peuvent garder en mémoire les contenus qui transitent. Chaque nœud peut également répondre aux requêtes s'il possède les contenus recherchés.

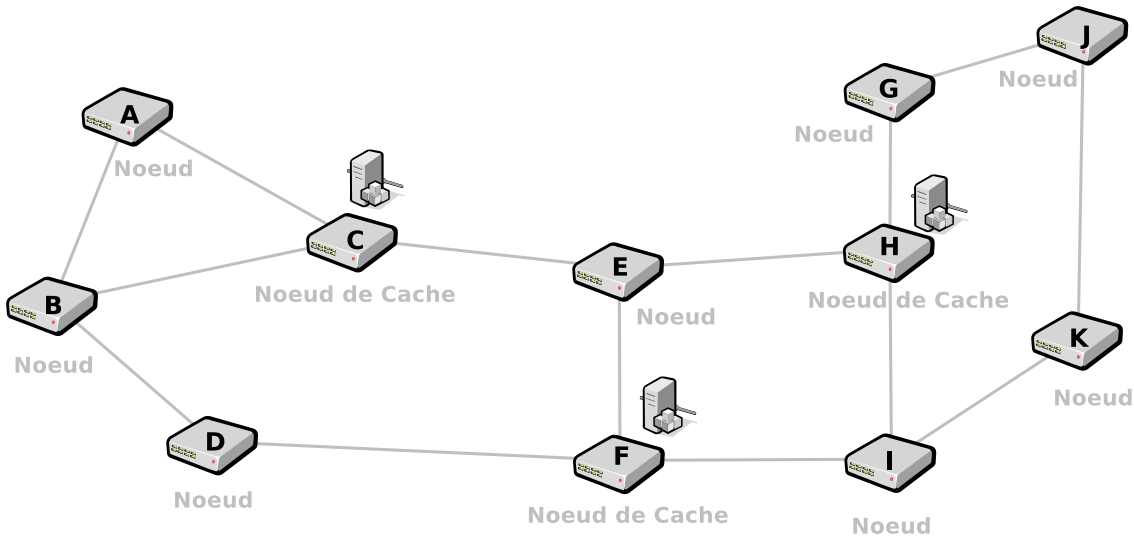


Figure 5.3 – Exemple de réseau NDN avec la topologie Abilene où seuls les nœuds C, F et H possèdent une capacité de cache (*Content Store*).

Pour les autres scénarios, seulement 20%, 40%, 50% ou 80% des nœuds possèdent un *Content Store*, afin d'évaluer l'impact des caches dans le réseau à différentes densités. Ces scénarios peuvent évoquer les réseaux P2P où un nombre non défini de nœuds contribue aux échanges des contenus dans le réseau.

Nous considérons aussi deux autres scénarios, dans lesquels un seul nœud du réseau possède une capacité de cache. Deux cas sont alors distingués : le cas où l'unique *Content Store* est placé aléatoirement sur un nœud du réseau (*scénario 1*) et le cas où il est placé sur le nœud le plus connecté (*scénario 1MC*). Ces deux scénarios rappellent l'utilisation de CDN (*Content Delivery Network*) dans l'Internet, où quelques serveurs seulement sont déployés pour aider à distribuer les contenus et réduire la charge sur le serveur d'origine.

5.2.3 Métriques

Nous évaluerons les performances de SRSC ainsi que de la technique de *flooding* NDN avec les trois métriques suivantes :

— le *Cache Hit Ratio*, qui indique l'efficacité du réseau et indique la proportion de requêtes ayant été satisfaites par les *Content Stores* des nœuds ;

— le nombre de messages *Interest* et *Data* envoyés, représentant le plan de données ;

— le nombre de messages de contrôle envoyés, représentant le plan de contrôle. Ce dernier est constitué des communications entre les nœuds et le contrôleur, permettant la mise en place de notre protocole de routage SRSC dans le réseau NDN. Il correspond au surcoût en communication généré par SRSC (*overhead*).

Pour les n nœuds du réseau, appelés n_i, notons I_i le nombre de requêtes (i.e. message *Interest*) reçues par n_i et D_i le nombre de réponses (i.e. message *Data*) retournées par le *Content Store* de n_i. Le *Cache Hit* (c_i) représente le pourcentage de requêtes satisfaites par le *Content Store* de n_i et se calcule ainsi :

$$\mathrm{c}_i = \frac{D_i}{I_i}$$

Le *Cache Hit Ratio* (CHR) total du réseau équivaut au *Cache Hit* de manière globale :

$$\mathrm{CHR} = \frac{\sum_{i=1}^{n} D_i}{\sum_{j=1}^{n} I_j}$$

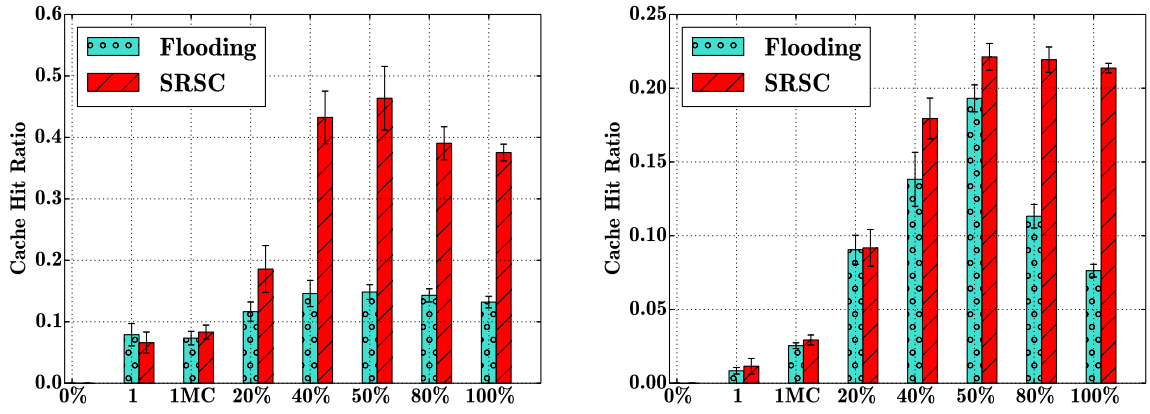
Comme il s'agit de simulations, nous n'avons pas évalué les temps de traitement des requêtes car ce sont des temps simulés dans des topologies via un outil de simulation. Nous utiliserons la métrique des temps de traitement lors de l'évaluation de l'implantation de notre protocole dans le chapitre suivant.

5.3 Résultats de l'évaluation

Nous évaluons SRSC en le comparant au *flooding*, méthode par défaut utilisée dans NDN sur deux topologies, Abilene et Geant, avec les paramètres de simulation présentés en Table 5.1. Nous mesurons les trois métriques décrites en Section 5.2.3 sur plusieurs scénarios où nous faisons varier le nombre de nœuds disposant de *Content Stores*, c'est à dire de nœuds pouvant stocker les contenus.

Cache Hit Ratio :

Le *Cache Hit Ratio* est la métrique la plus importante pour évaluer les architectures orientées contenu et notamment NDN car elle indique la capacité à trouver un contenu dans les caches du réseau. Nous présentons sur les Figures 5.4a et Figure 5.4b les performances de SRSC pour le *Cache Hit* sur les deux topologies d'opérateurs. L'axe des abscisses correspond à la proportion de nœuds équipés d'un *Content Store* (différents scénarios) et l'axe des ordonnées représente le *Cache Hit Ratio* dans le réseau.



(a) Abilene **(b)** Geant

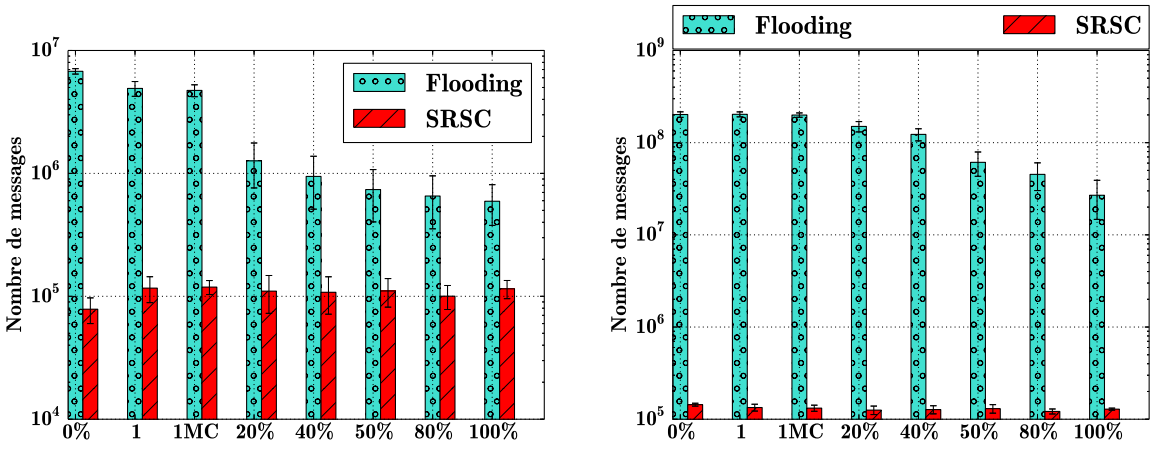
Figure 5.4 – *Cache Hit Ratio* obtenu lors de la comparaison entre SRSC et la méthode de *flooding*

Sur ces deux Figures, nous pouvons voir que quels que soient les topologies et les scénarios considérés, le protocole SRSC atteint un meilleur *Cache Hit* que la technique du *flooding*. Il y a cependant des différences de performances en fonction des scénarios et de la topologie.

Tout d'abord, avec le scénario 0%, aucune mémoire n'est déployée dans le réseau donc le *Cache Hit* sera forcément nul. Ensuite, avec les scénarios 1, 1MC, 20%, 40% et 50%, le *Cache Hit Ratio* augmente puis diminue au delà de ce seuil. On observe en fonction des scénarios que le *Cache Hit Ratio* augmente avec le nombre de *Content Stores* pour SRSC et le *flooding*, puis, au delà de 50% de *Content Stores*, les performances diminuent. Ce résultat est dû au fait que deux nœuds voisins vont posséder, à quelques changements près, les mêmes contenus en cache. Plus les mémoires caches seront éloignées les unes des autres, plus la chance de trouver des contenus variés sera grande. Déployer trop de mémoires cache oblige à aller, saut après saut, vérifier dans les *Content Stores* des nœuds, diminuant l'efficacité de ces derniers pour des contenus demandés peu populaires qui ne seront, par conséquent, pas gardés dans les caches.

Dans la topologie Abilene, pour le scénario 50%, SRSC atteint jusque 50% de performance de *Cache Hit*, ce qui veut dire qu'une requête sur deux sera résolue dans le réseau, ce qui est trois fois plus efficace que la technique de *flooding*. D'une façon plus générale, dès lors qu'on dépasse les 20% de nœuds équipées de mémoire cache, les performances avec SRSC en matière de *Cache Hit Ratio* sont largement supérieures au *flooding*.

Dans la topologie Geant, la différence de performance entre SRSC et le *flooding* est moins prononcée, avec un *Cache Hit* atteignant 0,22 pour le scénario 50% et des performances relativement similaires pour le *flooding* qui atteint 0,19. Pour les scénarios 1, 1MC, 20%, 40% et 50% les performances entre SRSC et le *flooding* sont également similaires et la différence moins prononcée que pour la topologie Abilene, bien que SRSC obtient tout de même de meilleures performances pour le *Cache Hit*. Toutefois, pour les scénarios 80% et 100%, les performances avec le *flooding* se dégradent considérablement et SRSC obtient des performances de cache jusqu'à trois fois supérieures au *flooding*.



(a) Abilene **(b)** Geant

Figure 5.5 – Nombre de message *Interest* émis pour SRSC et la méthode de *flooding*

Comme expliqué précédemment, les contenus non-populaires n'étant pas conservés dans les mémoires caches car ils sont remplacés en premier, augmenter le nombre de *Content Stores* va diminuer le *Cache Hit* lorsque ces contenus moins populaires seront demandés.

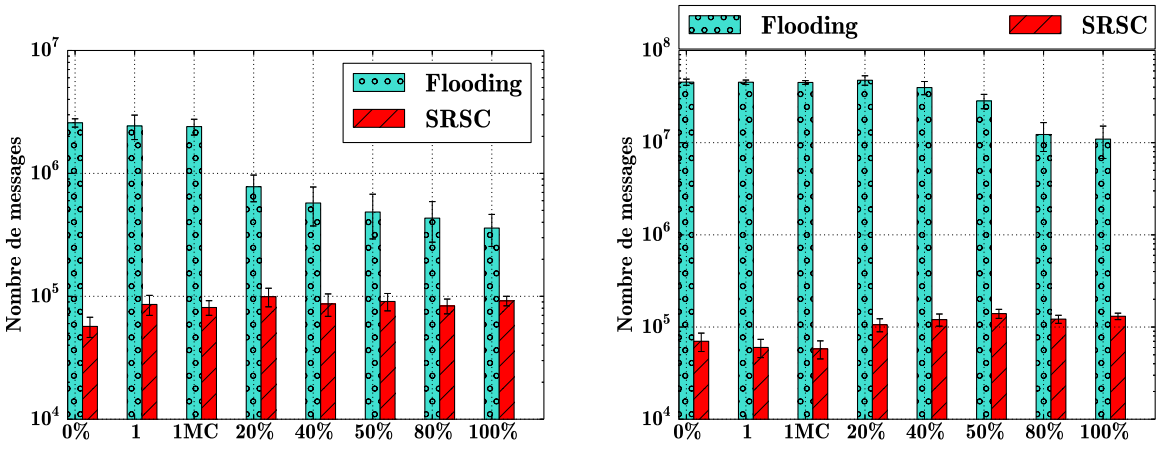
Ce résultat est d'autant plus visible sur la topologie Geant que sur Abilene car elle contient beaucoup plus de nœuds. Ces nœuds étant aussi bien plus connectés, le nombre de messages *Interest* transmis suite à l'inondation du réseau sera largement supérieur. La topologie Abilene possédant peu de nœuds, connectés de façon très linéaire, affiche des répercussions bien moins importantes en terme de *Cache Hit* suite à cette augmentation du nombre de *Content Stores*, car l'inondation y est bien moins coûteuse.

Finalement, nous observons lors de ces expériences qu'il existe un effet de seuil au delà duquel il n'est pas nécessaire d'augmenter le nombre de *Content Stores* pour augmenter le *Cache Hit Ratio* dans le réseau. En effet, cela n'augmentera pas les performances de cache de l'architecture, voire au contraire pourra les diminuer et notamment pour le *flooding* (*scénario 80%* et *scénario 100%*).

Les résultats montrent qu'en comparaison de la méthode de *flooding*, SRSC obtient de bien meilleures performances pour l'utilisation des caches dans le réseau, quel que soit le nombre de *Content Stores* déployés. En effet SRSC ne voit pas ses performances dégradées au delà de 50% de *Content Stores* comme c'est le cas avec le *flooding*, mais semble atteindre un seuil au delà duquel les performances stagnent.

Nombre de messages – Plan de données

Les figures 5.5a pour Abilene et 5.5b présentent le nombre de messages *Interest* pour les deux topologies tandis que les figures 5.6a et 5.6b présentent le nombre de messages *Data* transmis dans le réseau.



(a) Abilene **(b)** Geant

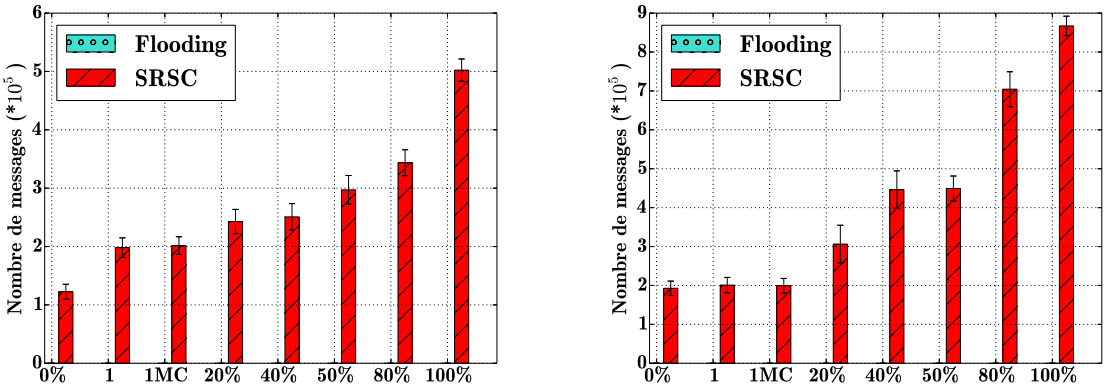
Figure 5.6 – Nombre de message *Data* émis pour SRSC et la méthode de *flooding*

Pour le *flooding*, dans les deux topologies, nous pouvons observer que le nombre de messages *Interest* et *Data* décroît lorsque la proportion de nœuds avec le nombre de *Content Stores* augmente (histogrammes turquoises). Par exemple, avec Abilene, le nombre de messages *Interest* est divisé par dix entre les *scénarios 0%* et *scénario 50%*, puis décroît ensuite plus lentement. Sur une topologie très linéaire comme Abilene, en plaçant quelques *Content Stores* seulement dans le réseau NDN on arrive à limiter la propagation des messages, provoqués par le *flooding*. Toutefois, le nombre de messages dans la topologie Abilene est d'un ordre de magnitude plus faible que dans la topologie Geant.

Avec Geant, le nombre de messages *Interest* et *Data* est similaire pour les scénarios 0%, 1 et 1MC, puis on observe une faible diminution en fonction des scénarios. Sur une topologie très connectée comme celle de Geant, il y a moins d'impact à déployer des *Content Stores* lorsque le *flooding* est utilisé. S'il y a peu de *Content Stores*, les messages *Interest* seront tout de même inondés avant d'atteindre une mémoire cache et les réponses *Data* seront également largement diffusées dans le réseau, augmentant considérablement le nombre de messages.

En revanche, avec SRSC, le nombre de messages *Interest* et *Data* (histogrammes rouges) est relativement constant pour chaque scénario et topologie. En effet, comme la requête est acheminée vers le cache le plus proche via consultation du contrôleur, une augmentation du nombre de *Content Stores* aura un impact bien moins important : cela raccourcira la transmission des requêtes de quelques sauts mais le nombre de messages va rester relativement constant puisque le routage dépend de la consultation du contrôleur.

De nos résultats, on observe que le protocole SRSC permet bien de limiter le nombre de messages émis dans le plan de données comparé à la traditionnelle méthode de *flooding*. Nous observons aussi que SRSC a un impact bien plus important dans la topologie Geant que dans Abilene sur la réduction du nombre de messages émis : on dénombre jusque 100 fois moins de messages pour Abilene contre jusque 1000 fois moins de messages émis pour Geant. Ceci vient du fait que la topologie Geant est plus connectée et plus vaste que Abilene. Dans cette topologie très connectée, tous les messages *Interest* seront plus largement dupliqués à chaque nœud ainsi que les réponses *Data* qui leur sont associées en comparaison de la topologie Abilene.



(a) Abilene **(b)** Geant

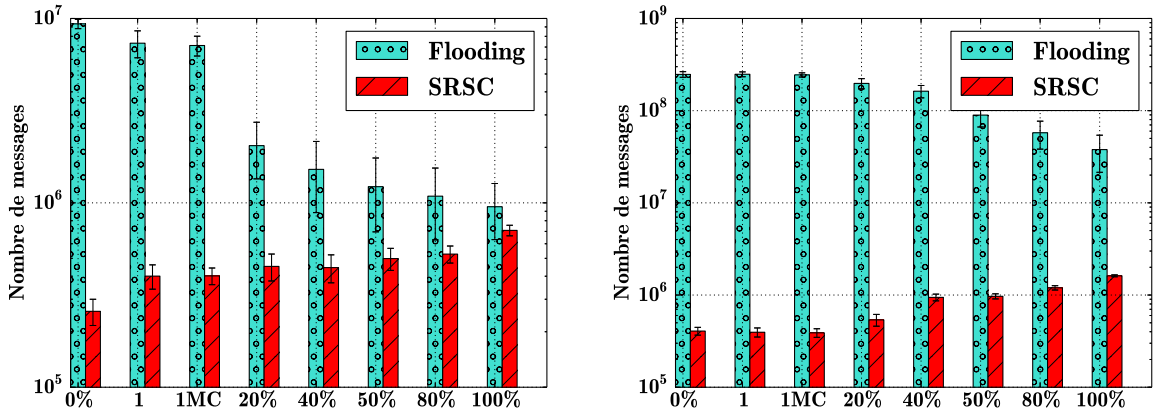
Figure 5.7 – Nombre de messages émis sur le plan de contrôle par SRSC

Nombre de messages – plan de contrôle

Nous nous consacrons maintenant à l'évaluation du nombre de messages de contrôle émis dans le réseau. Ces messages de contrôle sont les messages *Interest* et *Data* utilisés par SRSC pour permettre aux nœuds de communiquer avec le contrôleur et de diffuser les informations. Il s'agit de messages de signalisation spécifiques à SRSC ; bien entendu, la méthode de *flooding* n'en émet naturellement pas puisqu'elle n'utilise pas un tel plan de contrôle. Ces résultats sont présentés en Figure 5.7a pour la topologie Abilene, et Figure 5.7b pour la topologie Geant.

Pour les deux topologies, on observe que le nombre de messages de contrôle augmente avec le nombre de mémoires cache déployées dans le réseau. En effet, avec de nombreux caches, les contenus vont être plus largement stockés dans le réseau et chaque copie va engendrer un message d'avertissement au contrôleur pour indiquer que le nœud possède la copie. Le nombre de caches dans le réseau va ainsi avoir un impact sur le nombre de messages de contrôle générés.

Ces messages de signalisation générés par SRSC sont propres à ce protocole et n'existe pas avec le *flooding*. On peut alors se demander si le surcoût en communication (*overhead*) est trop important avec SRSC et il convient de mettre ces résultats en perspective avec la quantité totale de messages émis : plan de contrôle et plan de données cumulés. Le nombre de messages total est présenté sur les figures 5.8a et Figure 5.8b. Comme nous l'avions vu précédemment, SRSC génère significativement moins de messages que le *flooding* pour le plan de données. Pour ce qui est du nombre de messages total, si SRSC a ajouté un surcoût en communication (plan de contrôle), il génère toutefois beaucoup moins de messages en totalité que le *flooding* et se révèle donc plus efficace en nombre de messages transmis. Par exemple, SRSC génère jusque 100 fois moins de messages au total que le *flooding* pour le scénario 50% sur la topologie Abilene (Figure 5.8a). En effet, avec le *flooding*, la diffusion de tous les messages sur toutes les interfaces ainsi que les réponses associées génèrent un nombre de messages supplémentaires nettement plus important que pour SRSC. Comme vu en Chapitre 4, SRSC concentre les messages de signalisation vers le contrôleur et évite l'inondation de messages qui est une technique très coûteuse en utilisation des ressources du réseau. Le bénéfice avec SRSC est d'autant plus important dans les grandes topologies très connectées comme Geant, car lorsque la technique de *flooding* est mise en œuvre, de nombreux messages sont retransmis.



(a) Abilene **(b)** Geant

Figure 5.8 – Nombre de messages total émis lors de la comparaison entre SRSC et la méthode de *flooding* (plan de donnée et plan de contrôle cumulés)

Synthèse des résultats

Nous résumons ici les principales observations de notre évaluation via expériences de simulation pour comparer notre protocole SRSC avec le *flooding*, technique par défaut de NDN pour transmettre les contenus.

— Pour le *Cache Hit* (Figure 5.4), SRSC est plus performant que la technique du *flooding* et ce, quels que soient les topologies ou les scénarios considérés. Sur certaines configurations (Abilene, scénario 50%) les performances sont même doublées. Cela indique que notre protocole SRSC, via l'usage d'un contrôleur dans le réseau, est plus à même de récupérer les contenus dans les caches du réseau, ce qui est l'objectif principal pour un protocole de routage adapté à l'architecture NDN.

— SRSC génère moins de messages dans le réseau que la technique de *flooding* quels que soient les topologies ou scénarios considérés (Figures 5.8a et 5.8b). En plus d'être plus performant pour accéder au contenu, SRSC génère moins de message et sera plus apte à passer à l'échelle de l'Internet. C'est d'ailleurs d'autant plus vrai que la topologie est large et fortement connectée (cf : topologie Geant figure 5.8b et Abilene 5.8a). De plus, quand bien même SRSC représente un surcoût en communication via un plan de contrôle et des messages spécifiques pour échanger avec le contrôleur, ce nombre de messages reste relativement limité par rapport au nombre total de messages émis dans le réseau.

— Le nombre de *Content Stores* (caches) déployés dans le réseau a un impact sur les performances de l'architecture. Dans nos expériences de simulation, on observe qu'au delà de 50% de nœuds équipés de capacité de stockage, les performances ne sont plus fondamentalement améliorées. En effet, le *Cache Hit* atteint son maximum pour le scénario 50% et les scénarios avec plus de mémoires cache ne sont pas plus performants. Avec SRSC, Le nombre de messages ne varie pas de façon significative quels que soient les scénarios.

Ainsi, de tous ces résultats on peut affirmer que SRSC remplit ses objectifs d'être un protocole de routage adapté à l'architecture NDN. Il est à la fois plus performant pour le *Cache Hit* que la technique par défaut utilisée par NDN (*flooding*). Il permet également de générer moins de messages dans le réseau, ce qui le rend plus apte à passer le facteur d'échelle et notamment pour un déploiement pour un futur Internet.

De plus, nous observons un effet de seuil avec des performances maximales avec 50% de nœuds équipés de mémoires cache (*Cache Hit*). C'est un résultat important car cela montre que pour déployer une infrastructure comme NDN à très large échelle, voire à l'échelle de l'Internet, il n'est pas nécessaire de déployer de la mémoire cache sur tous les nœuds du réseau. Ces mémoires doivent être assez larges et sont généralement l'un des éléments les plus coûteux d'une infrastructure, cela indique qu'il est possible de déployer l'architecture NDN a un coût d'infrastructure plus faible (CAPEX) et ce, pour un niveau de performances supérieur.

Il est à noter que nous avons choisi d'évaluer SRSC avec la stratégie de gestion des caches par défaut de NDN (*Last Recently Used*). Les performances des caches pourraient être encore améliorées avec des stratégies de caches adaptées à NDN comme celles déjà proposées dans de précédents travaux [101, 52, 102].

5.4 Résumé

Au cours de ce chapitre, nous avons évalué à travers des expériences de simulation notre protocole SRSC et l'avons comparé à la technique par défaut utilisée dans NDN, le *flooding*. Nous avons effectué ces expériences en nous servant du simulateur NS-3 ainsi que son module ndnSIM. Notamment, nous avons évalué ces deux approches sur deux topologies d'opérateurs différentes Abilene et Geant ; nous avons également considéré des scénarios où tous les nœuds ne sont pas nécessairement équipés de capacité de stockage.

Les résultats obtenus valident notre approche puisque notre protocole SRSC est plus performant en matière de *Cache Hit*, c'est à dire qu'il permet d'accéder au contenu de façon plus efficace. De plus, il génère moins de messages que la technique de *flooding*, ce qui en fait un protocole qui pourrait être déployé à très large échelle car consommant moins de ressources réseau (bande passante, etc.).

Nous montrons aussi que le nombre de *Content Stores* déployés dans le réseau a un impact sur les performances de l'architecture et il n'est pas nécessaire que tous les nœuds en soient équipés.

Dans nos expériences, les performances sont maximales lorsque 50% des nœuds sont équipés de capacités de stockage. C'est un résultat important pour les opérateurs réseaux car cela montre que l'architecture NDN peut être déployée à un coût d'infrastructure moindre (CAPEX) avec le même niveau de performances, voire davantage. Au delà du coût d'infrastructure, cela aura également un impact sur le coût énergétique de l'architecture et des activités d'opérations sur celles-ci (OPEX).

Suite à ces expériences de simulation, nous avons implanté notre protocole SRSC dans NDNx, l'implantation de l'architecture *Named-Data Networking*. Nous présentons les résultats de nos expériences sur une plateforme expérimentale dans le chapitre suivant.

Chapitre 6

Implantation dans NDNx : plateforme expérimentale

Sommaire

6.1 Introduction 95
6.2 Plateforme expérimentale virtualisée 96
6.3 Évaluation de SRSC via notre plateforme expérimentale virtualisée 97
6.3.1 Démonstration de faisabilité de SRSC 97
6.3.2 Évaluation de SRSC et comparaison avec NLSR 100
6.4 Discussion 108
6.5 Résumé 109

6.1 Introduction

En parallèle de nos travaux, d'autres protocoles de routage ont été proposés et notamment NLSR [72], qui est devenu le protocole par défaut de NDNx, implantation de l'architecture NDN. Les expériences via simulations permettent de valider un prototype, mais pas d'évaluer un protocole dans des conditions réelles d'expérimentation. De plus, NLSR n'existant pas dans ndnSIM, il était impossible de le comparer par simulation avec notre protocole pour en voir les différences en termes de performances.

Ainsi, au delà des simulations, nous avons développé une plateforme expérimentale utilisant NDNx et y avons implanté notre protocole SRSC, rendant ainsi possible la comparaison des deux protocoles NLSR et SRSC.

Dans ce chapitre, nous commençons par décrire la plateforme expérimentale en Section 6.2, puis nous détaillons dans la Section 6.3 les expérimentations réalisées. Nous présentons une preuve de concept confirmant le fonctionnement de SRSC sur notre plateforme avant de comparer les performances de SRSC avec celles de NLSR. Nous terminons par une discussion des résultats obtenus suite à ces expérimentations.

6.2 Plateforme expérimentale virtualisée

Des plateformes pour évaluer l'architecture NDN existent déjà, mais elles ne permettent pas de modifier les composants principaux de NDN (comme le processus NFD de transmission des messages par exemple). Ces plateformes se contentent de donner la possibilité de tester des applications qui seront déployées au dessus de l'architecure NDN [103, 89]. D'autres nécessitent des ressources et du matériel spécifiques pour les déployer [104, 105], ce qui peut rendre leur installation complexe.

Dans l'optique d'avoir une plateforme simple à mettre en place et ne nécessitant aucune configuration ou matériel dédié, nous avons fait le choix d'utiliser la technologie Docker et une plateforme virtualisée [106]. Docker est un logiciel libre permettant d'automatiser le déploiement d'applications dans des conteneurs logiciels, ce qui répond parfaitement à nos attentes en termes de moyens et facilité de déploiement. Contrairement à des machines virtuelles classiques, un conteneur Docker n'inclut pas de système d'exploitation et s'appuie sur les fonctionnalités du système hôte fournies par l'infrastructure sous-jacente. Les conteneurs sont donc portables et peuvent fonctionner sur tout type d'infrastructure (Cloud, station de travail ou data-center). Il est aussi possible de déployer plusieurs conteneurs sur une machine hôte unique, reliés entre eux par des liens réseaux virtuels, permettant de construire facilement des infrastructures réseau dans le but de les évaluer.

La Figure 6.1 illustre notre plateforme expérimentale virtualisée avec Docker. Chaque nœud est représenté par un conteneur qui encapsule NDNx [107], l'implantation libre (*Open Source*) de NDN. Afin de prendre en compte SRSC nous avons modifié le processus gérant la transmission des messages, appelé NFD (*NDN Forwarding Daemon*). Les autres composants natifs de NDN tels que la PIT, la FIB ou le *Content Store* sont restés inchangés.

Chaque conteneur encapsule aussi les applications comme les clients (qui émettent les requêtes pour les contenus) et/ou les serveurs (fournissant les contenus). Les liens entre les nœuds du réseau sont représentés par des liens entre les faces des conteneurs. Les contrôleurs possèdent une implantation différente des traditionnels nœuds NDN, car ils doivent également gérer la topologie, calculer les chemins et localiser les contenus. Deux différents types de conteneurs ont alors été instanciés : l'un contenant notre implantation de NDNx/SRSC pour les nœuds du réseau (utilisant le processus NFD N) ; le second contenant celle du contrôleur (utilisant le processus NFD C).

Nous avons automatisé le déploiement de la plateforme via un script qui se charge de démarrer tous les contenus, instancier la topologie et les liens entre les nœuds, et modifier les paramètres expérimentaux tels que la taille des *Content Stores*, la taille du catalogue de contenu, le nombre de serveurs, et de clients. Au delà de l'évaluation de notre protocole de routage SRSC et comparaison avec NLSR, notre plateforme pourrait tout aussi bien être utilisée pour évaluer d'autres mécanismes conçus spécialement pour l'architecture NDN.

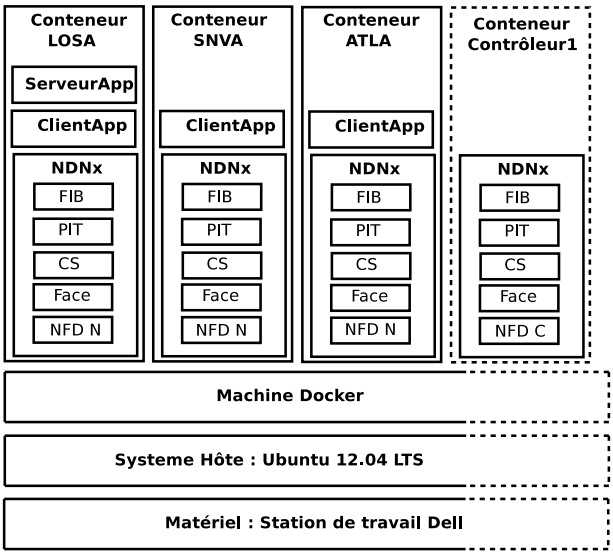


Figure 6.1 – Vision en pile de l'infrastructure Docker utilisée dans notre plateforme expérimentale

6.3 Évaluation de SRSC via notre plateforme expérimentale virtualisée

Afin d'évaluer l'implantation de notre protocole SRSC dans NDNx, nous avons mené plusieurs types d'expériences. Dans un premier temps, nous avons distingué plusieurs scénarios de routage possibles avec SRSC, suivant par exemple que les requêtes soient propagées directement vers le serveur d'origine ou vers une autre copie locale, disponible dans un *Content Store*. Ces différents scénarios de routage sont tous évalués indépendamment, comme une démonstration de faisabilité de notre protocole SRSC.

Ensuite, nous effectuons des expériences à plus large échelle avec un scénario plus réaliste (nombre d'utilisateurs, catalogue, nombre de requêtes etc.) Dans cette expérience, nous évaluerons les performances de SRSC et le comparerons à NLSR.

6.3.1 Démonstration de faisabilité de SRSC

L'architecture NDN repose sur un réseau de cache et le protocole SRSC a été conçu pour acheminer les requêtes vers la copie locale la plus proche (routage *anycast*). Dans cette expérience, nous testons les différents cas qui pourraient se produire pour l'acheminement d'une requête afin de démontrer la faisabilité de notre protocole et son aptitude à communiquer avec le contrôleur et permettre aux nœuds d'acheminer les requêtes.

Nous avons ainsi distingué quatre scénarios de routage distincts lorsqu'une requête arrive sur un nœud. Ces quatre scénarios sont présentés via l'exemple de la Figure 6.2 et décrits ci-après :

— Scénario de Routage Complet
— Scénario de Cache
— Scénario de Routage de Proximité
— Scénario de Transmission Normale

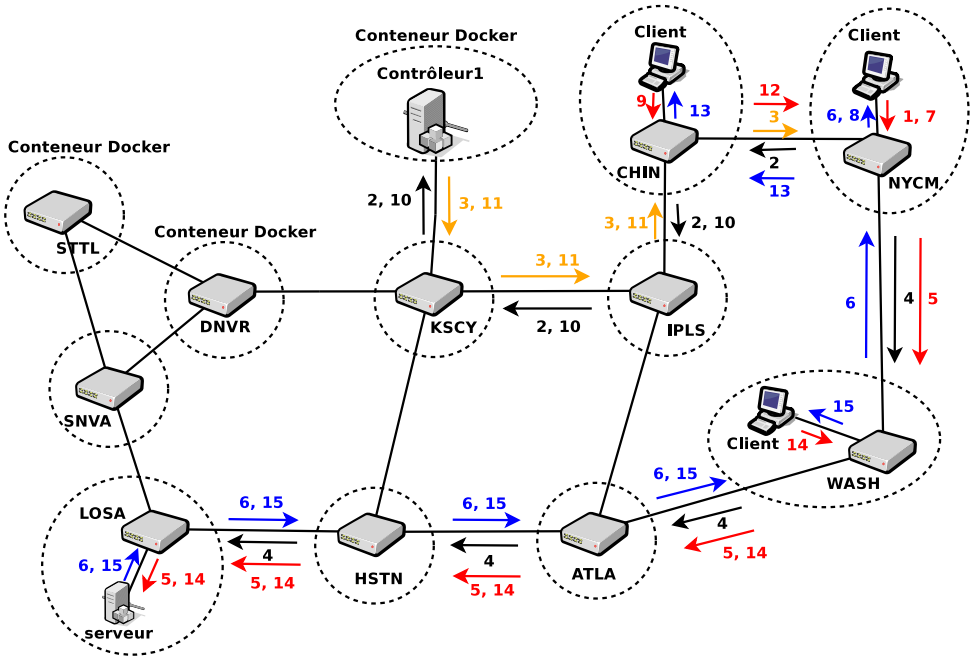


Figure 6.2 – Expérience sur la plateforme Docker : SRSC a été implémenté dans NDNx et déployé dans les conteneurs de la plateforme. Quatre scénarios de routage sont présentés. **Scénario de Routage Complet :** messages 1 à 6 ; **Scénario de Cache :** messages 6 et 7 ; **Scénario de Routage de Proximité :** messages 9 à 13 ; **Scénario de Transmission Normale :** messages 14 et 15

Scénarios de routage

Scénario de Routage Complet (SRC) : Dans ce scénario, lorsqu'un *Interest* arrive à un nœud et que celui-ci n'a ni le contenu dans son *Content Store*, ni les règles dans ses tables PIT ou FIB pour le transmettre, le nœud contacte le contrôleur pour lui demander la règle de *forwarding*. Lorsqu'il n'y a que le serveur d'origine qui possède une copie de la donnée, ou que le serveur d'origine est l'entité la plus proche du nœud d'où provient la requête, le contrôleur va indiquer d'acheminer la requête vers le serveur. Pour cela, il installe les règles dans la FIB du nœud ayant reçu la requête, qui va ensuite, grâce à l'envoi d'un message *InterestCreatePath*, installer les règles dans l'ensemble des nœuds sur le chemin. Ce scénario complet est le plus utilisé au démarrage car aucun contenu n'est encore présent en cache et aucune règle n'a encore été installée dans les nœuds.

Scénario de Routage de Proximité (SRP) : Dans le scénario SRP, le nœud ne possède pas non plus le contenu dans son *Content Store*, ni de règle dans les FIB ou PIT. Le nœud contacte donc le contrôleur, mais dans ce cas le contenu est situé dans le *Content Store* d'un nœud plus proche et non plus sur le serveur d'origine comme dans le scénario SRC. Le contrôleur envoie donc les règles pour transmettre le message *Interest* jusqu'à ce nœud. Dans ce scénario, le serveur ne sera pas sollicité ce qui diminuera sa charge, tout en améliorant les performances pour l'utilisateur qui récupère le contenu à proximité.

Scénario de Transmission Normale (STN) : Dans ce scénario, un nœud qui reçoit un message possède la règle de transmission dans sa FIB pour acheminer le message *Interest* vers une copie locale. Il n'a donc pas besoin de contacter son contrôleur pour transmettre la requête.

Pour les scénarios SRC et SRP, une fois que le premier nœud a contacté le contrôleur pour obtenir la règle, les prochains nœuds du chemin suivront ce scénario et n'auront pas besoin de contacter le contrôleur à chaque saut. En effet, le message *InterestPathCreate* aura été envoyé pour installer les règles dans les autres nœuds du chemin.

Scénario de Cache (SC) : Dans ce scénario, lorsqu'un nœud reçoit un message *Interest*, il possède le contenu dans son *Content Store* et peut répondre directement à la requête. C'est le cas idéal dans l'architecture NDN, où une copie du contenu est trouvée dans un des caches du réseau. Ce cas se produit quand des contenus ont été diffusés dans le réseau et sont stockés plus proches des utilisateurs.

Les quatre scénarios de routage représentent les différents cas pouvant se produire lors du routage avec SRSC. Comme il s'agit d'une démonstration de faisabilité du protocole SRSC, cette expérience a été réalisée sur la topologie Abilene décrite dans la Figure 6.2. L'application serveur et son catalogue de fichiers sont placés sur le nœud LOSA, les clients sont placés sur les nœuds CHIN, NYCM et WASH et le contrôleur est connecté au nœud KCSY pour être au cœur du réseau. Le nom des nœuds correspond à la ville américaine où ils se situent pour la topologie Abilene. Par exemple LOSA pour Los Angeles ou encore WASH pour Washington. Les envois de messages *Interest* (rouge), *Data* (Bleu), *Interest* de contrôle (noir) et *Data* de contrôle (jaune) sont représentés sur le schéma pour chacun des différents scénarios. Pour chaque scénario, une unique requête est envoyée.

Résultats

Les résultats de cette expérience sont présentés sur la Figure 6.3. Chaque histogramme donne les durées d'exécution dans chaque entité du réseau pour chacun de ces quatre scénarios. Ces durées d'exécution peuvent provenir de trois différentes entités : (i) le contrôleur (rouge) ; (ii) les nœuds (gris) ; (iii) le serveur (bleu). Ces résultats indiquent clairement qu'en fonction des scénarios, les temps de calcul sont variés et présentent des performances différentes.

Tout d'abord, si l'on se réfère au scénario SC, le délai d'exécution est très faible et notamment en comparaison avec les autres scénarios de routage. En effet, dans cette configuration, le nœud dispose du contenu en cache et n'a besoin ni du contrôleur, ni du serveur, ni même de transmettre son message *Interet* dans le réseau et d'attendre le message *Data* correspondant. Ce délai correspond uniquement au délai pour récupérer la donnée dans le *Content Store* du nœud et la transmettre.

Les scénarios SRP et STN présentent, quant à eux, des temps de traitement similaires mais avec des répartitions différentes. Avec STN, le contrôleur n'ayant pas besoin d'être contacté, le délai est composé principalement du temps de traitement de la requête par le serveur. À cela s'ajoutent des délais d'exécution des nœuds du réseau qui ont transmis la requête. Avec SRP, le

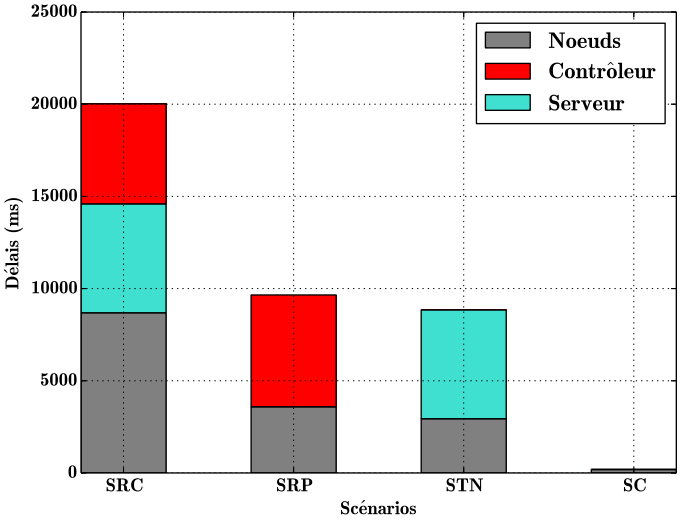


Figure 6.3 – Démonstration de faisabilité – évaluation des quatre scénarios de routage avec SRSC

nœud doit contacter le contrôleur pour obtenir les règles de routage, qui le dirige vers le nœud le plus proche. La majeure partie du délai correspond alors au temps de calcul des plus courts chemins par le contrôleur.

Le scénario SRC présente le plus important délai ; car le nœud doit tout à la fois contacter le contrôleur, le serveur, et tous les nœuds nécessaires à la transmission de la requête jusqu'au serveur d'origine.

Cette première expérience démontre que notre protocole fonctionne comme il se doit et sait s'adapter aux différentes situations, que des règles soient connues ou non, ou que le contenu soit présent en cache ou non. On peut noter que les délais sur un contrôleur comprennent également le temps de calcul du plus court chemin via l'algorithme de Djikstra. Une fois les chemins stockés dans la mémoire du contrôleur, ces calculs ne seront plus nécessairement effectués, réduisant par la même le délai d'exécution dans ce type de scénario pour de futures requêtes du réseau.

Une fois cette première étape validée, nous évaluons les performances de SRSC avec un scénario plus réaliste et à plus large échelle et nous le comparons au protocole de routage NLSR utilisé par NDNx.

6.3.2 Évaluation de SRSC et comparaison avec NLSR

Dans cette expérience, nous évaluons les performances de l'implantation de notre protocole SRSC et son déploiement au travers notre plateforme expérimentale virtualisée, présentée en Section 2 de ce chapitre. Nous comparons également ces performances à NLSR, que nous avons également déployé sur notre plateforme.

Comme nous l'avions expliqué Chapitre 3, NLSR est un protocole de routage pour NDN dans lequel les nœuds conservent les chemins vers les serveurs d'origine. Ainsi il ne permet pas de faire du routage *anycast*, c'est-à-dire de router vers la copie locale la plus proche. Il nécessite également de nombreuses annonces augmentant le trafic de signalisation.

Au cours de cette expérience, afin de vérifier l'impact du nombre de *Content Stores* présents dans le réseau, nous utiliserons deux types de configuration : (i) tous les nœuds possèdent une capacité de stockage ; (ii) uniquement 50% des nœuds possèdent une capacité de stockage. En effet, lors de l'évaluation de performances via simulation (Chapitre 5), les meilleures performances en matière de *Cache Hit* et même de signalisation étaient atteintes avec 50% de nœuds équipés de mémoires cache. Dans cette configuration, les nœuds équipés d'un *Content Store* ont été choisis aléatoirement à chaque lancement du scénario.

Les mémoire caches des nœuds sont homogènes, c'est à dire de même taille. Les paramètres utilisés pour ces expériences sont les mêmes que ceux du Chapitre 5 et sont rappelés ci-après.

Paramètres expérimentaux

Les évaluations des performances ont été effectuées sur les topologies Abilene et Geant présentées Chapitre 5 et que nous avons instanciées sur notre plateforme expérimentale virtualisée.

Le catalogue de fichiers du serveur contient 10 000 contenus. Chaque fichier a une taille variable entre 1 et 1024 octets. Nous avons placé un client sur chacun des nœuds de la topologie (11 pour Abilene et 41 pour Geant), chaque client émettant 10 000 requêtes. Ces requêtes suivent une loi de probabilité ZipF (avec le paramètre α=1.1) pour déterminer le contenu demandé. Lorsque les nœuds en sont équipés, chaque *Content Store* a une capacité de 100 contenus (0,1% du catalogue) et utilise les stratégies LCD *(Leave Copy Down)* pour que les nœuds puissent décider quel contenu stocker, et LFU (*Least Frequently Used*) pour décider lequel remplacer quand la mémoire est pleine. Ces deux stratégies sont celles utilisées par défaut dans NDN et consistent à stocker tous les contenus qui passent par le nœud, et remplacer les contenus les moins souvent utilisés en cas de manque d'espace.

Les messages *Data* ont une taille de 1024 octets et peuvent donc transporter tous les fichiers en une seule fois.

Les métriques prises en compte dans cette évaluation sont le *Cache Hit Ratio* total du réseau, le temps de traitement des requêtes dans les entités du réseau, le nombre de messages pour le plan de données (*Interest* et *Data*), ainsi que le nombre de messages pour le plan de contrôle (*Interest* et *Data* spécifiques de SRSC).

Pour chaque configuration (tous les nœuds équipés de *Content Store*, ou 50% d'entre eux), nous avons effectué 10 fois chaque expérience et présenterons la moyenne obtenue ainsi que l'écart type. Tous ces paramètres sont reportés dans la Table 6.1.

Résultats expérimentaux

L'objectif de ces expériences est d'évaluer les performances de SRSC, et de les comparer avec NLSR dans le même environnement et avec les mêmes paramètres et configuration.

Paramètres	Valeurs
Topologies	Abilene, Geant (ISP)
Taille du Catalogue	10 000 contenus
Nombre de Clients	11 ; 41
Nombre de Requêtes par Client	10 000.
Modèle de Popularité	Zipf(α=1.1)
Taille des Caches	100 contenus (0,1% du catalogue)
Stratégie de Mise en Cache	LCD
Politique de Remplacement des Caches	LFU
Taille d'un message Data	1024 octets

Table 6.1 – Paramètres expérimentaux

Configuration	Valeur	Écart-type
SRSC 50%	45.9	0.06
SRSC 100%	47	0.002
NLSR 50%	27.9	0.05
NLSR 100%	50	0.007

(a) Abilene

Configuration	Valeur	Écart-type
SRSC 50%	55.77	1.42
SRSC 100%	43.9	1.79
NLSR 50%	35.27	2.09
NLSR 100%	45.14	1.97

(b) Geant

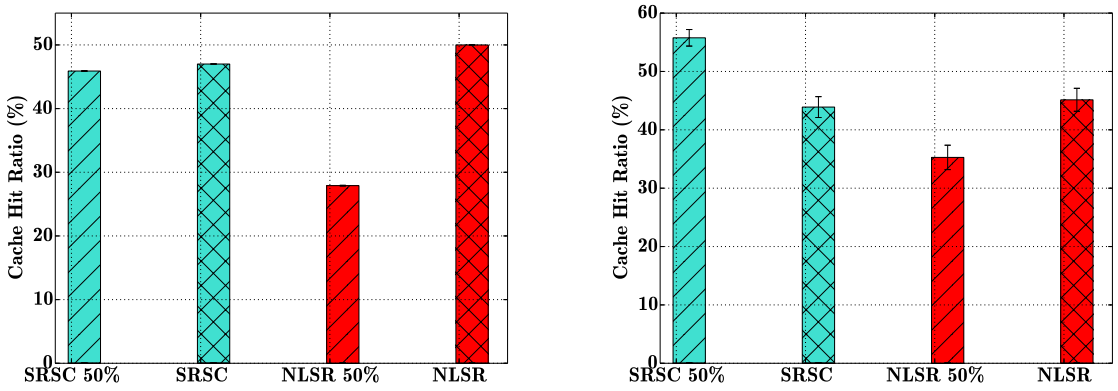
Table 6.2 – *Cache Hit Ratio* pour SRSC et NLSR sur la plateforme expérimentale

***Cache Hit Ratio* :** Les performances en matière de *Cache Hit Ratio* sont présentées sur la figure 6.4 et récapitulées en Table 6.2. Lorsque tous les nœuds sont équipés de *Content Stores*, sur les deux topologies les performances sont similaires avec un léger avantage pour NLSR (cf. Table 6.2).

En revanche, les observations sont différentes pour une configuration avec 50% de nœuds équipés de *Content Stores*. Dans la topologie Abilene, les performances de SRSC sont stables quel que soit le nombre de *Content Stores* déployés. Les performances de NLSR sont quant à elles, diminuées de moitié (cf. Table 6.2a). Sur la topologie Geant, le *Cache Hit Ratio* de SRSC augmente avec 50% de mémoire cache, tandis qu'il diminue pour NLSR (cf. Table 6.2b).

Les performances de SRSC sont ainsi meilleures avec moins de nœuds équipés de mémoire cache. SRSC présente également de meilleurs performances sur une topologie comme celle de Geant qui est très connectée.

NLSR ne semble pas adapté à des configurations où tous les nœuds ne possèdent pas de *Content Stores*. En effet, NLSR permet de récupérer une copie dans le chemin vers le serveur



(a) Abilene **(b)** Geant

Figure 6.4 – *Cache Hit Ratio* pour SRSC et NLSR, évalué sur la plateforme expérimentale

d'origine (*in-path*), alors que SRSC permet de faire du routage *anycast* vers la copie disponible la plus proche. Si le nombre de mémoires cache diminue, SRSC sait s'adapter avec sa vision centralisée grâce au contrôleur et trouver une alternative. Cela serait même bénéfique compte tenu de l'augmentation de ses performances en matière de *Cache Hit*.

Ces observations corroborent celles du Chapitre 5 et le fait que les performances de l'architecture NDN ne diminuent pas nécessairement lorsqu'il y a moins de *Content Stores* déployés et qu'un protocole de routage adapté comme SRSC est utilisé.

Ce cas illustré en Figure 6.5, où un utilisateur, connecté au Nœud1, souhaite accéder à un contenu disponible dans le Serveur d'origine ainsi que dans le *Content Store* du Nœud4.

Avec NLSR, lorsque l'utilisateur émet la requête (message 1), celle ci est envoyée vers le serveur (messages 2, 3 et 4) générant trois MISS (i.e. trois accès mémoire dans le *Content Store* sans trouver la donnée demandée) car le contenu n'est pas disponible dans les caches qui se trouvent sur le chemin. Arrivée au serveur, la requête trouve le contenu, générant un HIT (i.e. donnée trouvée dans le *Content Store*), et la donnée est retournée avec un message *Data* message (5 à 8), générant donc un *Cache Hit Ratio* de 0.25 (un HIT pour trois MISS).

Avec le protocole SRSC, lorsque l'utilisateur émet sa requête (message 1), celle-ci génère un MISS au Nœud1 et ce dernier contacte son Contrôleur pour trouver le contenu le plus proche (messages 2 et 3). Le Contrôleur répond avec le chemin pour accéder au Nœud4 car il sait que c'est la copie la plus proche (message 4 et 5). Le Nœud1 émet donc le message *Interest* vers le Nœud4, générant un HIT car la donnée est dans son *Content Store* (message 6). Elle est ensuite retournée vers l'utilisateur (message 7 et 8). Dans cet exemple, le *Cache Hit* Ratio est de 0.5 (un HIT pour un MISS), montrant que SRSC a su mieux prendre en compte des différents paramètres du réseau (topologie, copies, etc.).

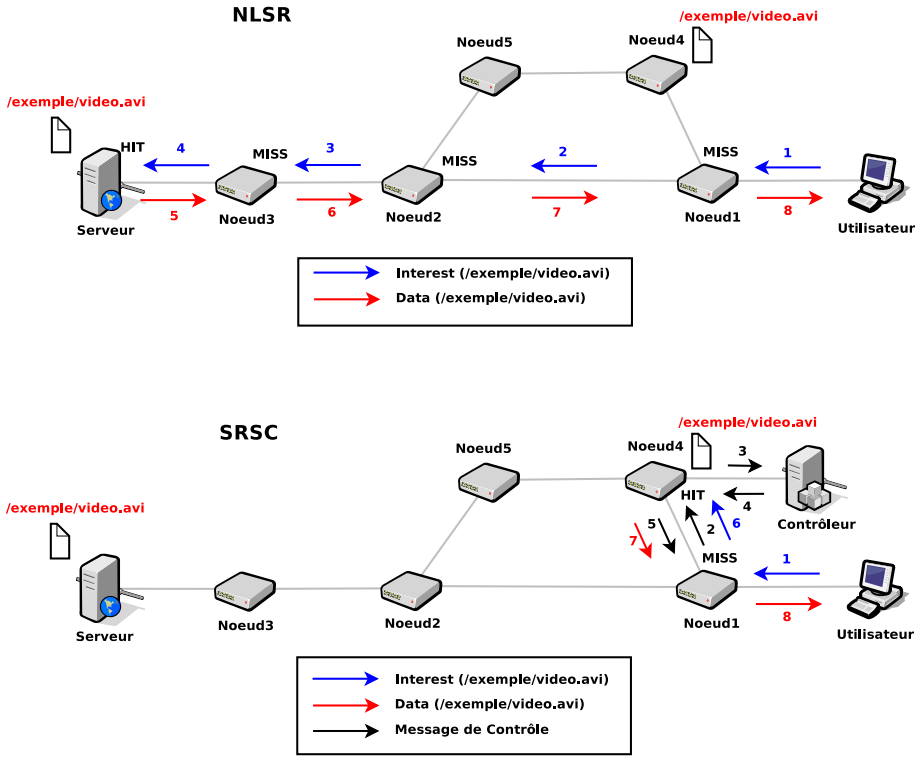
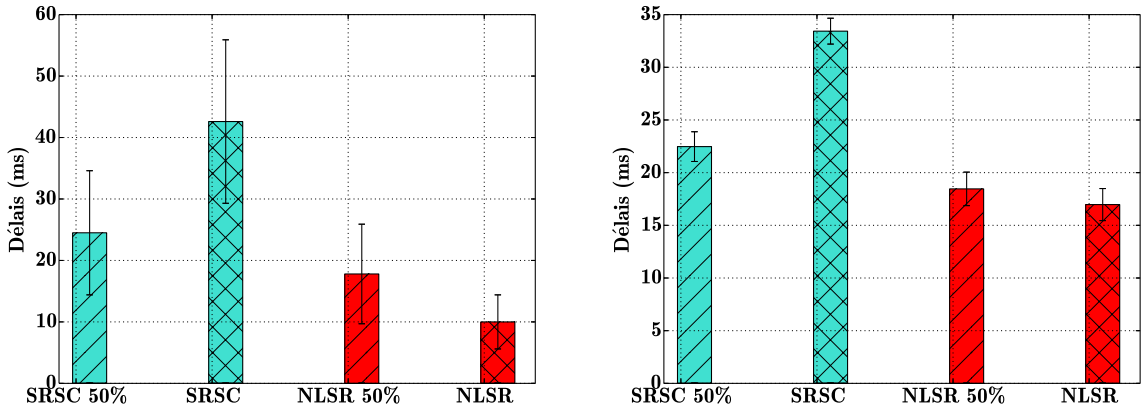


Figure 6.5 – Exemple de calcul de *Cache Hit Ratio* avec SRSC et NLSR



(a) Abilene **(b)** Geant

Figure 6.6 – Temps de traitement pour SRSC et NLSR, évalué sur la plateforme expérimentale

Configuration	Valeur	Écart-type
SRSC 50%	24.5	10.1
SRSC 100%	42.6	13.3
NLSR 50%	17.8	8.1
NLSR 100%	10	47

(a) Abilene

Configuration	Valeur	Écart-type
SRSC 50%	22.47	1.41
SRSC 100%	33.43	1.23
NLSR 50%	18.46	1.59
NLSR 100%	16.97	1.52

(b) Geant

Table 6.3 – Temps de traitement pour SRSC et NLSR sur la plateforme expérimentale

Temps de traitement : Les temps de traitement sont présentés sur la Figure 6.6 et récapitulés en Table 6.3. Sur les deux topologies, NLSR présente des temps de traitement plus courts que SRSC. Toutefois, le fait de diminuer le nombre de *Content Stores* a eu un impact sur les deux protocoles mais a produit des effets inverses.

En effet, pour NLSR sur la topologie Abilene (cf : Table 6.3a), ce temps a quasiment doublé lorsque 50% des nœuds n'ont plus de *Content Store* alors qu'il est presque divisé par deux avec SRSC (topologie Abilene). Dans la topologie Geant (Figure 6.6b), la diminution du nombre de *Content Store* a un impact moins important pour NLSR (cf Table 6.3b) mais garde la même tendance, à savoir, moins de mémoires cache dans le réseau augmenterait le temps de traitement de NLSR. En revanche, pour SRSC, la diminution de ce temps est non négligeable entre les deux configurations (cf : 6.3a).

On peut également noter que les temps de traitement obtenus avec SRSC sont moins importants sur la topologie Geant que sur Abilene, contrairement à NLSR. La différence entre ces deux protocoles vient du fait que la topologie Geant dispose d'un plus grand nombre de nœuds et de liens qui permettent au contrôleur de proposer d'autres chemins alternatifs plus courts (*anycast*). En revanche, pour NLSR, le chemin entre le client et le serveur d'origine étant généralement plus long dans de grandes topologies, les requêtes vont traverser plus de nœuds et NLSR présentera des temps de traitement plus longs.

Configuration	Valeur	Écart-type
SRSC 50%	412 700	33 800
SRSC 100%	296 951	1 736
NLSR 50%	348 313	57 522
NLSR 100%	263 290	7 827

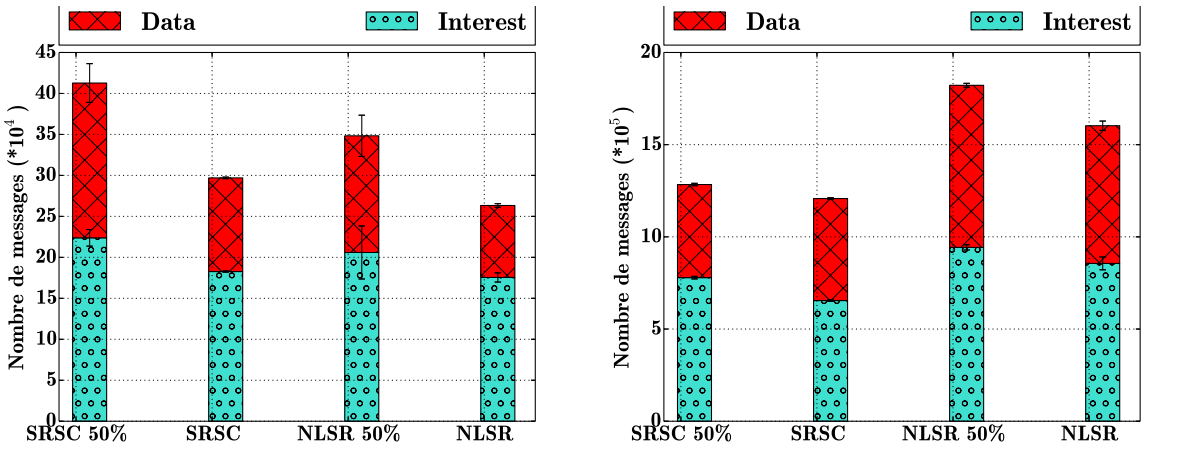
(a) Abilene

Configuration	Valeur	Écart-type
SRSC 50%	1 283 732.4	13 877.92
SRSC 100%	1 309 023	9 823.26
NLSR 50%	1 821 988.66	24 959.61
NLSR 100%	1 602 834.2	59 963.08

(b) Geant

Table 6.4 – Plan de données (*Interest* et *Data*) pour SRSC et NLSR sur la plateforme expérimentale

Nombre de messages et surcharge en communication :



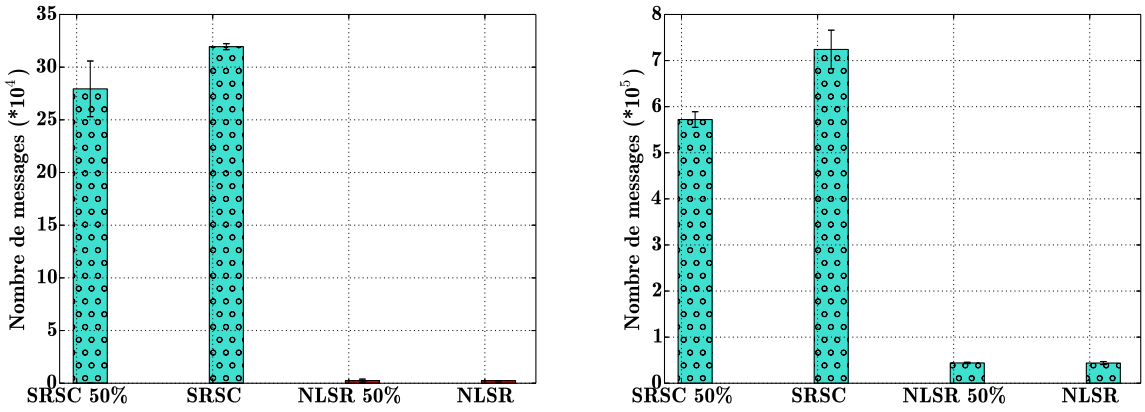
(a) Abilene **(b)** Geant

Figure 6.7 – Nombre de messages *Interest* et *Data* pour SRSC et NLSR, évalué sur la plateforme expérimentale

Plan de données : La Figure 6.7 présente le nombre de messages du plan de données pour ces deux protocoles (*Interest* et *Data*), et les valeurs sont présentées Table 6.4. Tout d'abord, si l'on observe les deux topologies ensemble, pour NLSR et SRSC, on note que le nombre de messages *Interest* et *Data* augmente sensiblement pour la configuration avec 50% de nœuds équipés de mémoires cache.

Pour SRSC, cette observation est légèrement différente des expériences de simulations où les messages du plan de données (*Interest* et *Data*) restaient relativement constants en fonction du nombre de mémoires (cf. Figure 5.5 et Figure 5.6 du Chapitre 5). Les pertes de messages qui peuvent se produire avec une réelle implantation déployée sur une plateforme expérimentale peuvent expliquer cette distinction de l'environnement simulé. Les écarts type de la configuration 50% sont plus importants, ce qui indique que leurs différents placements aléatoires, sur chacune des dix expériences, va avoir un effet sur le nombre de messages envoyés.

Si l'on distingue les deux topologies, SRSC génère plus de messages que NLSR sur la topologie Abilene, mais génère moins de messages que NLSR sur la topologie Geant. Une topologie fortement connectée convient mieux à SRSC, car des chemins plus courts peuvent être obtenus, limitant le nombre de retransmissions de messages.



(a) Abilene **(b)** Geant

Figure 6.8 – Nombre de messages de contrôle émis pour SRSC et NLSR, évalué sur la plateforme expérimentale

Configuration	Valeurs	Écart-type
SRSC 50%	279 389	26 444
SRSC 100%	319 428	2 853
NLSR 50%	2 495	1 410
NLSR 100%	2 399	108

(a) Abilene

Configuration	Valeurs	Écart-type
SRSC 50%	572 054.6	16 947.23
SRSC 100%	724 158.25	41 729.41
NLSR 50%	44 012.3	1 483.3
NLSR 100%	43 810.6	3 386.95

(b) Geant

Table 6.5 – Plan de contrôle pour SRSC et NLSR sur la plateforme expérimentale

Plan de contrôle : Le nombre de messages du plan de contrôle est présenté sur la Figure 6.8 et résumé en Table 6.5. On observe immédiatement que le nombre de messages de signalisation est bien plus important pour SRSC que pour NLSR. En regardant plus en détail, on observe cependant que le nombre de messages diminue pour SRSC entre la configuration standard et la configuration avec 50% de *Content Stores*. C'est différent pour NLSR qui voit son nombre de messages augmenter.

De plus, le nombre de messages dans la topologie Geant (cf. Table 6.5b) est plus important pour les deux protocoles et configurations que pour la topologie Abilene (cf : Table 6.5a). Cependant, le nombre de messages a "uniquement" doublé pour SRSC dans Geant, alors que le nombre de messages a été démultiplié par un facteur 20 pour NLSR.

Ce résultat montre à nouveau que la topologie a un impact sur le comportement des protocoles. SRSC est plus performant sur des topologies très connectées alors que des topologies comme Abilene conviennent mieux à NLSR.

6.4 Discussion

La Figure 6.9 met en perspective le nombre de messages du plan de données et du plan de contrôle. On remarque que dans la topologie Abilene (Figure 6.9a), le plan de contrôle équivaut à environ la moitié des messages émis pour SRSC, mais à une infime partie pour NLSR. Dans la topologie Geant (Figure 6.9b), la proportion du plan de contrôle diminue pour SRSC alors qu'elle a sensiblement augmenté pour NLSR.

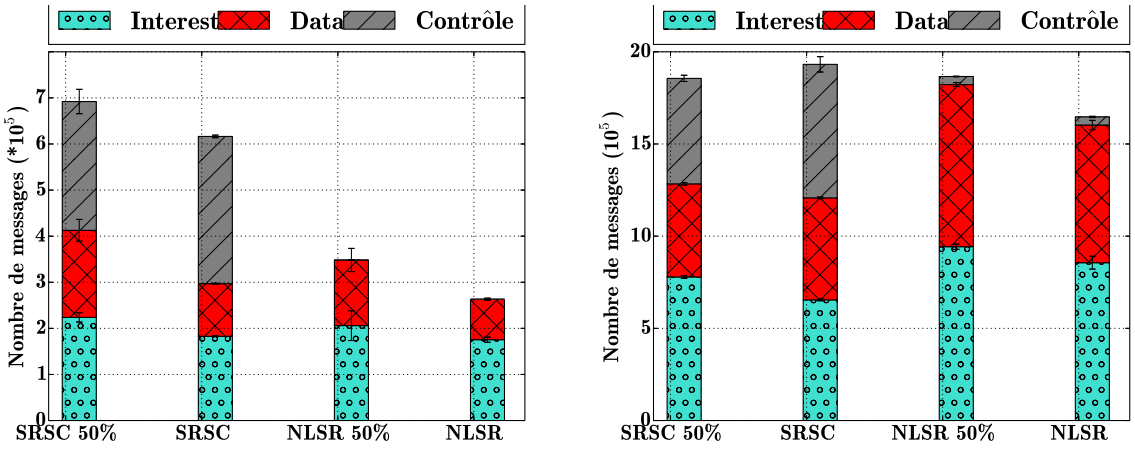
Il est cependant important de rappeler ici le fonctionnement de NLSR. Lorsqu'un nouveau contenu est ajouté, NLSR doit inonder le réseau de mises à jour pour que chaque nœud ait connaissance de cette donnée. Notre protocole SRSC ne nécessite qu'un seul message du nœud avertissant le contrôleur du nouveau contenu. En effet, SRSC fait émettre un message *Interest* pour chaque contenu disponible dans le serveur d'origine et son catalogue, ou à chaque nouvelle mise en cache dans un nœud du réseau, à chaque demande de route et pour l'installation des règles de transmission dans les nœuds. NLSR quant à lui, se contente de propager la route vers le préfixe général dans tout le réseau et ne recommence cette étape que lorsqu'un nouveau contenu est publié. Dans notre évaluation, NLSR n'a besoin d'annoncer qu'une seule fois le préfixe du catalogue sur le serveur et comme aucun nouveau contenu n'est ajouté, ce type d'annonce ne se reproduit pas dans nos expériences.

De plus, NLSR propage le préfixe général pour atteindre le serveur, alors que SRSC annonce chaque contenu ; c'est d'ailleurs ce qui va permettre à SRSC d'effectuer un routage *anycast*, là où NLSR ne peut effectuer que du routage vers le serveur d'origine (*in-path*).

Ainsi, nos expériences permettent bien d'évaluer tous les messages de signalisation de SRSC, mais ne permettent pas de mesurer tous les messages qui seraient propagés par NLSR dans un scénario plus réaliste avec des nouveaux contenus qui apparaîtraient continuellement dans le réseau.

Pour des raisons de puissance de calcul, il ne nous a pas été possible d'effectuer des évaluations sur des topologies beaucoup plus grandes, comparables à un Internet, ou regroupant plusieurs topologies d'opérateurs comprenant des milliers de nœuds. Avec des scénarios plus réalistes, ainsi que de nouveaux contenus publiés continuellement dans le réseau, au fur et à mesure de l'expérience comme c'est le cas dans l'Internet, les annonces par inondation de NLSR devraient générer un nombre bien plus important de messages qui pourrait dépasser largement SRSC. Cette inondation ne va pas avoir uniquement un impact sur le nombre de messages émis, mais aussi sur les temps de traitement et le passage à l'échelle d'une telle approche. Il était donc attendu que la signalisation de SRSC soit plus importante que celle de NLSR pour ces expériences

Il est nécessaire d'ajouter que nous avons déployé SRSC tel un prototype et que l'une des pistes d'amélioration sera, bien entendu, de réduire le nombre de messages de contrôle émis. C'est notamment possible en agrégeant les messages de contrôle et en associant SRSC à des stratégies de gestion des caches plus adaptées, limitant les remplacements de contenus dans les mémoires et donc, les transmissions de messages d'avertissement.



(a) Abilene **(b)** Geant

Figure 6.9 – Nombre de messages total (plan de donnée et plan de contrôle) pour SRSC et NLSR, évalué sur la plateforme expérimentale

6.5 Résumé

Dans ce chapitre, nous avons présenté l'évaluation de notre protocole SRSC sur notre plateforme expérimentale virtualisée, réalisée avec Docker.

Nous avons tout d'abord démontré la faisabilité de notre protocole de routage via une première expérience visant à obtenir les temps de traitement de chacun des scénarios de routage possibles avec SRSC. Nous avons ensuite évalué les performances de SRSC et l'avons comparé au protocole de routage NLSR à travers une expérience à plus large échelle.

En matière de *Cache Hit*, c'est à dire l'aptitude à trouver les contenus en cache, SRSC obtient des performances comparables à celle de NLSR. De plus, lorsqu'il y a moins de nœuds équipés de mémoire dans le réseau, SRSC présente même de meilleures performances que NLSR. Cela est dû à sa capacité à effectuer du routage *anycast* vers la copie locale la plus proche, tandis que NLSR effectue du routage *in-path* vers le serveur d'origine. À nouveau, c'est un résultat intéressant car cela indique que l'infrastructure NDN pourrait être déployée à des coûts moindres sans toutes les mémoires caches. Les temps de traitement sont cependant plus longs avec SRSC mais diminuent lorsque l'infrastructure comprend moins de mémoire, alors que ces temps augmentent avec NLSR. De même, le nombre de messages est plus important avec SRSC, mais il tend à diminuer avec moins de mémoire, et surtout sur des topologies très connectées comme celles de l'Internet.

Avec tous ces résultats d'expériences, nous pouvons conclure que SRSC remplit bien les objectifs que nous nous étions fixés, à savoir, proposer un nouveau protocole de routage adapté à l'architecture NDN. SRSC est une étape importante vers le déploiement de NDN à double titre : d'une part, il s'affranchit totalement de la pile protocolaire TCP/IP, frein au déploiement de NDN en tant qu'architecture à la base d'un Internet du futur. D'autre part, il permet d'effectuer du routage *anycast* et d'acheminer les requêtes vers les copies locales les plus proches, c'est à dire de profiter pleinement du réseau de cache que met en œuvre l'architecture NDN.

Chapitre 7

Conclusion

Sommaire

7.1 Résumé des contributions 112

7.2 Travaux futurs 112

Le réseau mondial qu'est l'Internet ne cesse de se développer et accueille chaque jour de nouveaux utilisateurs. Ces utilisateurs publient et s'échangent de nouveaux contenus continuellement. Les services de vidéos mis en place pour satisfaire les besoins des utilisateurs (vidéos à la demande, plateforme de *streaming* comme Youtube, réseau P2P type BitTorrent) génèrent la majeure partie du trafic total mondial.

Utilisée au commencement pour l'échange de mails et le web, l'architecture TCP/IP n'a pas été conçue pour délivrer cette immense quantité de contenu aux utilisateurs. Bien que l'infrastructure ait évolué au fil des années, l'architecture de l'Internet basée sur le protocole IP est restée sensiblement la même.

Dans le but de permettre à l'Internet d'assurer son nouveau rôle, les réseaux orientés contenu (ICN) ont été proposés. En se focalisant sur les données plutôt que sur les hôtes du réseau, les ICN ont pour but d'acheminer les données plus efficacement aux utilisateurs.

Plusieurs architectures ICN sont proposées et l'architecture NDN est, selon nous, la plus avancée d'entre elle, et rassemble la plus grande communauté de chercheurs et d'industriels pour son développement. En dotant ses nœuds d'une capacité de cache, le réseau NDN permet de stocker des copies des contenus dans le réseau pour améliorer leur distribution. Nous avons proposé dans cette thèse de doctorat un nouveau protocole de routage adapté à l'architecture NDN.

Dans ce chapitre, nous résumons nos contributions et présentons les perspectives futures pour la suite de nos travaux. Enfin nous listons les publications réalisées durant cette thèse de doctorat.

7.1 Résumé des contributions

Nous avons tout d'abord mené une étude sur les différentes architectures orientées contenu et plus spécifiquement sur les mécanismes de routage proposés dans ces architectures. Nous avons ainsi mis en évidence qu'aucune solution de routage existante ne permet d'effectuer un routage *anycast* de manière efficace. En effet, le routage vers la source la plus proche était une fonctionnalité mentionnée dans les documents décrivant les concepts de l'architecture NDN, mais il n'avait jamais été clairement défini ni mis en œuvre.

Afin de pallier ce manque, nous avons proposé le protocole de routage SRSC, qui repose sur le paradigme des réseaux logiciels, ou *Software-Defined Networking* (SDN) pour mettre en place un plan de contrôle dans NDN. le contrôleur SDN centralise les informations sur le réseau, comme la topologie, l'emplacement des serveurs ou les données dans les mémoires cache. Grâce à cela, le contrôleur peut installer les règles de transmission dans les nœuds pour permettre d'acheminer les requêtes vers le contenu le plus proche. Notre protocole SRSC offre la possibilité de déployer plusieurs contrôleurs dans un réseau NDN, afin d'assurer le partage de charge entre ces contrôleurs et, par là même, la gestion d'un réseau de grande envergure.

Nous avons ensuite évalué SRSC, à travers des expériences de simulation, en le comparant à la méthode d'inondation (*flooding*), utilisée par défaut par l'architecture NDN (Chapitre 5). Notre protocole présente de meilleures performances en matière d'utilisation des caches (*Cache Hit Ratio*) et de la signalisation. De plus, afin d'étudier l'impact des *Content Stores* sur les performances de l'architecture NDN, nous avons fait varier le nombre de nœuds NDN possédant un *Content Store*. Ces scénarios ont démontré qu'en diminuant le nombre de mémoires cache dans le réseau, l'architecture NDN reste performante avec SRSC, alors que ses performances sont fortement dégradées avec la méthode de *flooding*.

Suite à ces résultats positifs obtenus via un environnement de simulation, nous avons développé SRSC dans NDNx, l'implantation "libre" de l'architecture NDN (Chapitre 6). Nous avons également déployé une plateforme d'expérimentation virtuelle sous Docker pour évaluer notre protocole SRSC, et le comparer au protocole de routage NLSR, utilisé par défaut dans NDNx. SRSC exploite mieux les caches (augmentation du *Cache Hit Ratio*) lorsque moins de nœuds sont équipés de capacité de stockage (*Content Stores*) car il permet un routage *anycast*. En revanche, SRSC génère un nombre plus important de messages de contrôle et ce point est une piste d'amélioration du protocole pour la suite, même si cette observation peut être mitigée suivant la taille ou degré de connectivité de la topologie sous-jacente.

7.2 Travaux futurs

Nos travaux présentent de nombreuses perspectives qu'il conviendrait de poursuivre. Tout d'abord, le contrôleur pourrait être étendu de sorte à proposer de nouvelles fonctionnalités dans le réseau NDN. Il pourrait par exemple permettre une politique de routage à la demande grâce à ses capacités à superviser le trafic. Une requête pourrait alors être envoyée sur un chemin particulier pour réguler le trafic, ou vers les caches de plus grande capacité. Le contrôleur pourrait également assurer des fonctions de sécurité : de par sa connaissance du réseau, il pourrait servir

de relais de confiance entre les nœuds, ou d'autorité de certification. Il pourrait également ajouter des règles sur des nœuds pour rejeter le trafic (*drop*), c'est-à-dire mettre en œuvre des pare-feux à la volée et spécifiques à certaines requêtes.

Concernant notre protocole, une intégration dans la distribution NDNx pourrait être envisagée. Pour cette intégration, SRSC devrait être amélioré afin de générer moins de messages de signalisation et notamment, en agrégeant les messages de contrôle. De plus, une comparaison avec le protocole CRoS-NDN qui utilise aussi le paradigme SDN pourrait aussi être envisagée, mais il n'y a, à notre connaissance à ce jour, aucune implantation de CRoS-NDN.

Enfin, une perspective intéressante serait la mise en œuvre et le déploiement à très large échelle de l'architecture NDN. C'est d'ailleurs l'un des buts du projet français ANR DOCTOR [108] dans lequel ces travaux se sont inscrits. Pour cela, le projet Doctor a proposé une passerelle HTTP/NDN (Gateway), permettant de transmettre les datagrammes IP de l'Internet sur un réseau NDN et récripoquement. Cette passerelle va permettre d'intégrer des réseaux NDN à l'Internet et permettre son déploiement progressif.

Une question ouverte au sujet de l'architecture NDN concerne le nommage des données. En effet, de nombreux travaux dans la communauté NDN ont déjà porté sur les mécanismes de gestion des caches, le contrôle de congestion, le routage, ou la sécurité. Il conviendrait de proposer un nouveau système de certification du nommage adapté aux architectures orientées contenu, qui garantirait l'unicité des noms.

Listes des publications

— Large-Scale Experiments of SRSC : A Forwarding Scheme for NDN.
E. Aubry et T. Silverston.
To be submitted.

— Implementation and Evaluation of a Controller-based Forwarding Scheme for NDN.
E. Aubry, T. Silverston et I. Chrisment.
In AINA 2017-The 31st IEEE International Conference on Advanced Information Networking and Applications. Taipei, Mars 2017.

— Deployement of SRSC forwarding Scheme in NDN.
E. Aubry et T. Silverston.
In IC 2016-Internet Conference - poster session. Tokyo, Septembre 2016

— Croissance Verte dans NDN: Déploiement des Content Stores.
E. Aubry, T. Silverston et I. Chrisment.
In ALGOTEL 2016-18èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications. Bayonne, Mai 2016.

— Green growth in NDN: Deployment of content stores.
E. Aubry, T. Silverston et I. Chrisment.
In LANMAN 2016- IEEE Local and Metropolitan Area Networks. Roma , Juin 2016

— PRCS: protocole de routage pour CCN basé sur SDN.
E. Aubry, T. Silverston et I. Chrisment.
In ALGOTEL 2015—17èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications. Beaune, Juin 2015.

— SRSC: SDN-based routing scheme for CCN.
E. Aubry, T. Silverston et I. Chrisment.
In NetSoft 2015-1st IEEE Conference on Network Softwarization. London, Avril 2015

Autres publications

— CrowdOut: A mobile crowdsourcing service for road safety in digital cities.
E. Aubry, T. SILVERSTON, A. Lahmadi et O. Festor.
In PERCOM 2014- IEEE Pervasive Computing and Communications Workshops. Budapest, Mars 2014

— CrowdOut: un service de crowdsourcing pour la sécurité routière dans les villes numériques.
E. Aubry, T. SILVERSTON, A. Lahmadi et O. Festor.
In Ubimob 2013-9èmes journées francophones Mobilité et Ubiquité. Nancy, Juin 2013

Table des figures

Le numéro de page où trouver la figure est indiqué à droite de chaque ligne.

2.1 Évolution de la quantité de trafic sur Internet entre 1984 et 2014 [1] 14
2.2 Exemple de liaison entre plusieurs AS, exécutant les protocoles de routage internes (IS-IS, OSPF et RIP) et externe (BGP) 16
2.3 Exemple de hiérarchie des serveurs de DNS 17
2.4 Principe de fonctionnement du protocole IP à travers un exemple 18
2.5 Principe de fonctionnement des serveurs de DNS [35] 18
2.6 Principe de fonctionnement du multicast 19
2.7 Principe de fonctionnement des réseaux de distribution de contenus 21
2.8 Principe de fonctionnement du réseau Pair-à-Pair pour l'échange de fichier 22
2.9 Principe de fonctionnement du réseau Pair-à-Pair avec le système BitTorrent [4] . 22
2.10 Corps des messages *Interest* et *Data* utilisés dans NDN 24
2.11 Schéma des composants du nœud NDN [50] 25
2.12 Architecture CCN/NDN 26
2.13 Architecture DONA 27
2.14 Architecture PURSUIT 29
2.15 Architecture NetInf 30

3.1 Exemple d'utilisation du schéma de routage adaptatif 39
3.2 Exemple de fonctionnement du protocole RTR permettant de transmettre les messages *Interest* vers les *Content Stores* 40
3.3 Principe de fonctionnement du schéma de *forwarding* utilisant les fonctions de hachage, en mode Asymétrique ; permettant de retourner la réponse par le plus court chemin 43
3.4 Échanges de messages effectués entre les composants d'un nœuds lors de la transmission d'un *Interest* avec le protocole OSPFN 45
3.5 Illustration du mode de fonctionnement de l'approche de routage BFR, basée sur les filtres de Bloom. 47
3.6 Illustration du mode de fonctionnement du schéma de routage DCR, utilisant la distance pour émettre les requêtes. 49
3.7 Mode de routage du protocole SCAN permettant l'envoi vers les multiples copies du réseau dans le but d'améliorer le transfert des données. 50

3.8 Gestion du nœud NDN par le *resource manager*, permettant la gestion des ressources et la mise en place de politiques de routage spécifiques 51
3.9 Illustration du mode de fonctionnement du schéma de routage CRoS-NDN, utilisant un contrôleur fourni par le paradigme des réseaux logiciels. 52

4.1 Plan de routage SDN pour architecture NDN. 63
4.2 Topologie du réseau utilisée dans cet exemple 68
4.3 *Bootstrapping* : Étape 1 70
4.4 *Bootstrapping* : Étape 2 70
4.5 *Bootstrapping* : Étape 3 71
4.6 *Bootstrapping* : Étape 4 72
4.7 *Bootstrapping* : Étape 5 73
4.8 *Bootstrapping* : Étape 6 74
4.9 *Bootstrapping* : Étape 7 74
4.10 *Rules Forwarding* : Étape 1 78
4.11 *Rules Forwarding* : Étape 2 79
4.12 *Rules Forwarding* : Étape 3 80
4.13 *Rules Forwarding* : Étape 4 81

5.1 Représentation de la topologie Abilene déployée au États-Unis et comprenant 11 nœuds [99] 85
5.2 Représentation de la topologie Geant déployée en Europe et comprenant 41 nœuds [100] 85
5.3 Exemple de réseau NDN avec la topologie Abilene où seuls les nœuds C, F et H possèdent une capacité de cache (*Content Store*). 87
5.4 *Cache Hit Ratio* obtenu lors de la comparaison entre SRSC et la méthode de *flooding* 89
5.5 Nombre de message *Interest* émis pour SRSC et la méthode de *flooding* 90
5.6 Nombre de message *Data* émis pour SRSC et la méthode de *flooding* 91
5.7 Nombre de messages émis sur le plan de contrôle par SRSC 92
5.8 Nombre de messages total émis lors de la comparaison entre SRSC et la méthode de *flooding* (plan de donnée et plan de contrôle cumulés) 93

6.1 Vision en pile de l'infrastructure Docker utilisée dans notre plateforme expérimentale 97
6.2 Expérience sur la plateforme Docker : SRSC a été implémenté dans NDNx et déployé dans les conteneurs de la plateforme. Quatre scénarios de routage sont présentés. **Scénario de Routage Complet :** messages 1 à 6 ; **Scénario de Cache :** messages 6 et 7 ; **Scénario de Routage de Proximité :** messages 9 à 13 ; **Scénario de Transmission Normale :** messages 14 et 15 98
6.3 Démonstration de faisabilité – évaluation des quatre scénarios de routage avec SRSC 100
6.4 *Cache Hit Ratio* pour SRSC et NLSR, évalué sur la plateforme expérimentale . . 103
6.5 Exemple de calcul de *Cache Hit Ratio* avec SRSC et NLSR 104
6.6 Temps de traitement pour SRSC et NLSR, évalué sur la plateforme expérimentale 104
6.7 Nombre de messages *Interest* et *Data* pour SRSC et NLSR, évalué sur la plateforme expérimentale 106

6.8 Nombre de messages de contrôle émis pour SRSC et NLSR, évalué sur la plateforme expérimentale 107
6.9 Nombre de messages total (plan de donnée et plan de contrôle) pour SRSC et NLSR, évalué sur la plateforme expérimentale 109

Liste des tableaux

2.1 Comparaison des principales architectures ICN proposées 34

3.1 Comparaison des différents schémas de *forwarding* et de routage pour les ICNs . 54

4.1 Exemple de table de topologie présente dans les contrôleurs SDN. Ici le Contrôleur1 de la Figure 4.1 possède quatre nœuds, reliés par des liens de même coût (1 dans cet exemple) 64
4.2 Exemple de table de bordure présente dans les contrôleurs SDN. Ici le Contrôleur1 de la Figure 4.1 possède un seul nœud de bordure permettant de joindre un autre domaine, le Nœud5 lié au Contrôleur2. 64
4.3 Exemple de table de contenus présente dans les contrôleurs SDN. Ici le Contrôleur1 de la Figure 4.1 possède deux contenus différents, le premier disponible dans les nœuds Nœud2 et Noeud4, le second disponible dans Nœud3 65
4.4 Exemple de table de chemins présente dans les contrôleurs SDN. Ici le Contrôleur1 de la Figure 4.1 possède les chemins du Nœud2 au Nœud4, ainsi que du Nœud1 au Nœud3 65
4.5 *Interest* de Contrôle de SRSC défini pour le canal de communication entre les nœuds et les contrôleurs. Les *Data* de contrôle ont les mêmes préfixes et sont utilisés pour transporter des informations si nécessaire. 66

5.1 Paramètres de simulation 86

6.1 Paramètres expérimentaux 102
6.2 *Cache Hit Ratio* pour SRSC et NLSR sur la plateforme expérimentale 102
6.3 Temps de traitement pour SRSC et NLSR sur la plateforme expérimentale 105
6.4 Plan de données (*Interest* et *Data*) pour SRSC et NLSR sur la plateforme expérimentale 105
6.5 Plan de contrôle pour SRSC et NLSR sur la plateforme expérimentale 107

Bibliographie

[1] CISCO. The history and future of internet traffic. `https://blogs.cisco.com/sp/the-history-and-future-of-internet-traffic`.

[2] Napster. http ://www.napster.com/, 2006.

[3] Oliver Heckmann, Axel Bock, Andreas Mauthe, and Ralf Steinmetz. The edonkey file-sharing network. In Peter Dadam and Manfred Reichert, editors, *GI Jahrestagung (2)*, volume 51 of *LNI*, pages 224–228. GI, 2004.

[4] Bram Cohen. Incentives build robustness in bittorrent, 2003.

[5] Jean Burgess and Joshua Green. *YouTube : Online Video and Participatory Culture*. Polity, June 2009.

[6] www.dailymotion.com.

[7] www.netflix.com.

[8] CISCO. Cisco vni. `http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html`.

[9] Govind P. Agrawal. *Introduction*, pages 1–23. John Wiley Sons, Inc., 2011.

[10] Charles E. Perkins, Sherman R. Alpert, and Bobby Woolf. *Mobile IP ; Design Principles and Practices*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1997.

[11] James F. Kurose and Keith W. Ross. *Computer Networking : A Top-Down Approach (6th Edition)*. Pearson, 6th edition, 2012.

[12] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition, 2002.

[13] Douglas E. Comer. *Computer Networks and Internets*. Prentice Hall Press, Upper Saddle River, NJ, USA, 5th edition, 2008.

[14] Transmission Control Protocol. RFC 793, September 1981.

[15] Bob Quinn and Dr. Kevin C. Almeroth. IP Multicast Applications : Challenges and Solutions. RFC 3170, September 2001.

[16] Xuemin Shen, Heather Yu, John Buford, and Mursalin Akon. *Handbook of Peer-to-Peer Networking*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[17] Rajkumar Buyya, Mukaddim Pathan, and Athena Vakali. *Content delivery networks*, volume 9. Springer Science & Business Media, 2008.

[18] D. Perino and M. Varvello. A reality check for content centric networking. In *ACM ICN SIGCOMM WKSHPS 2011*, ICN '11, pages 44–49, New York, NY, USA, 2011. ACM.

[19] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(4) :181–192, August 2007.

[20] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, KC Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang. Named data networking. *ACM SIGCOMM 2014*, 44(3) :66–73, July 2014.

[21] Nikos Fotiou, Pekka Nikander, Dirk Trossen, and George C. Polyzos. *Developing Information Networking Further : From PSIRP to PURSUIT*, pages 1–13. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[22] Christian Dannewitz, Dirk Kutscher, BöRje Ohlman, Stephen Farrell, Bengt Ahlgren, and Holger Karl. Network of information (netinf) - an information-centric networking architecture. *Comput. Commun.*, 36(7) :721–735, April 2013.

[23] Mohamed Al-Ibrahim and Anton Cerny. Authentication of anycast communication. *Computer Network Security*, pages 419–423, 2003.

[24] Internet Engineering Task Force. *RFC 791 Internet Protocol - DARPA Inernet Programm, Protocol Specification*, September 1981.

[25] V Vetriselvan, Pravin R Patil, and M Mahendran. Survey on the rip, ospf, eigrp routing protocols. 2014.

[26] Jeff Doyle. *OSPF and IS-IS : Choosing an IGP for Large-Scale Networks*. Addison-Wesley Professional, 2005.

[27] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1) :269–271, December 1959.

[28] C. L. Hedrick. Routing information protocol, 1988.

[29] Uyless Black. *IP Routing Protocols : RIP, OSPF, BGP, PNNI and Cisco Routing Protocols*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.

[30] Richard Bellman. On a Routing Problem. *Quarterly of Applied Mathematics*, 16 :87–90, 1958.

[31] Y. Rekhter and T. Li. A border gateway protocol 4 (bgp-4), 1995.

[32] Paul V. Mockapetris and Kevin J. Dunlap. Development of the domain name system. In *In Proc. ACM SIGCOMM*, pages 123–133, 1998.

[33] Internet corporation for assigned names and numbers.

[34] Edward Levinson. Content-ID and Message-ID Uniform Resource Locators. RFC 2392, August 1998.

[35] James F. Kurose and Keith Ross. *Computer Networking : A Top-Down Approach Featuring the Internet*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[36] B. Fenner, H. He, B. Haberman, and H. Sandick. Internet Group Management Protocol (IGMP) / Multicast Listener Discovery (MLD)-Based Multicast Forwarding ("IGMP/MLD Proxying"). RFC 4605 (Proposed Standard), August 2006.

[37] Stephen Deering, Deborah Estrin, Dino Farinacci, Van Jacobson, Ching-Gung Liu, Liming Wei, Stephen Deering (xerox, Deborah Estrin (usc, Dino Farinacci (cisco, Van Jacobson (lbl, Ching gung Liu (usc, and Liming Wei (usc. Protocol independent multicast (pim), sparse mode protocol specification, 1994.

[38] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the ip multicast service and architecture. *IEEE Network*, 14(1) :78–88, Jan 2000.

[39] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the ip multicast service and architecture. *IEEE Network*, 14(1) :78–88, Jan 2000.

[40] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. Analyzing the performance of an anycast cdn. In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, pages 531–537, New York, NY, USA, 2015. ACM.

[41] I CloudFlare. Cloudflare-compare plans, 2013.

[42] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network : A platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3) :2–19, August 2010.

[43] Matei Ripeanu. Peer-to-peer architecture case study : Gnutella network. 2001.

[44] Petar Maymounkov and David Mazières. Kademlia : A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.

[45] M. Amadeo, C. Campolo, J. Quevedo, D. Corujo, A. Molinaro, A. Iera, R. L. Aguiar, and A. V. Vasilakos. Information-centric networking for the internet of things : challenges and opportunities. *IEEE Network*, 30(2) :92–100, March 2016.

[46] Named-Data Networking group. Named-Data Networking. `https://named-data.net/`.

[47] Xerox Company. Palo alto research center. `http://www.parc.com/`.

[48] FAQ Named-Data Project :. `https://named-data.net/project/faq/#How_does_NDN_differ_from_Content-Centric_Networking_CCN`.

[49] T. Berners-Lee. RFC 2396 : Uniform Resource Identifiers (URI). Technical report, MIT, 1998.

[50] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. *ACM CoNEXT '09*, 2009.

[51] Wei Koong Chai, Diliang He, Ioannis Psaras, and George Pavlou. Cache "less for more" in information-centric networks. In *Proceedings of the 11th International IFIP TC 6 Conference on Networking - Volume Part I*, IFIP'12, pages 27–40, Berlin, Heidelberg, 2012. Springer-Verlag.

[52] C. Bernardini, T. Silverston, and O. Festor. Socially-aware caching strategy for content centric networking. In *Networking Conference, 2014 IFIP*, pages 1–9, June 2014.

[53] J. Ren, W. Qi, C. Westphal, J. Wang, K. Lu, S. Liu, and S. Wang. Magic : A distributed max-gain in-network caching strategy in information-centric networks. In *IEEE INFOCOM WKSHPS 2014*, pages 470–475, April 2014.

[54] M. Rezazad and Y.C. Tay. Ccndns : A strategy for spreading content and decoupling ndn caches. In *IFIP Networking 2015*, pages 1–9, May 2015.

[55] I. Psaras, W. K. Chai, and G. Pavlou. In-network cache management and resource allocation for information-centric networks. *IEEE Transactions on Parallel and Distributed Systems 2014*, 25(11) :2920–2931, Nov 2014.

[56] D.R. Cheriton and M. Gritter. Triad : A new next-generation internet architecture, 2000.

[57] Vladimir Dimitrov and Ventzislav Koptchev. Psirp project – publish-subscribe internet routing paradigm : New ideas for future internet. In *Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies*, CompSysTech '10, pages 167–171, New York, NY, USA, 2010. ACM.

[58] European Union's Research and Innovation funding programme. Fp7. `https://ec.europa.eu/research/fp7/index_en.cfm`.

[59] Phani Raj Tadimety. *OSPF : A Network Routing Protocol*. Apress, Berkely, CA, USA, 1st edition, 2015.

[60] RISE. Scalable and adaptive internet solutions. `https://www.sics.se/projects/scalable-and-adaptive-internet-solutions`.

[61] FP-7. Architecture and applications of green information centric networkingr. `http://www.greenicn.org/`.

[62] G. García, A. Beben, F. J. Ramón, A. Maeso, I. Psaras, G. Pavlou, N. Wang, J. Śliwiński, S. Spirou, S. Soursos, and E. Hadjioannou. Comet : Content mediator architecture for content-aware networks. In *Future Network Mobile Summit (FutureNetw), 2011*, pages 1–8, June 2011.

[63] Giovanna Carofiglio, Massimo Gallo, and Luca Muscariello. Optimal multipath congestion control and request forwarding in information-centric networks. *Comput. Netw.*, 110(C) :104–117, December 2016.

[64] ICNRG. Information-Centric Networking Research Group. `https://irtf.org/icnrg`.

[65] Cheng Yi, Jerald Abraham, Alexander Afanasyev, Lan Wang, Beichuan Zhang, and Lixia Zhang. On the role of routing in Named Data Networking. Technical Report NDN-0016, NDN, December 2013.

[66] H. Yuan, T. Song, and P. Crowley. Scalable ndn forwarding : Concepts, issues and principles. In *2012 21st International Conference on Computer Communications and Networks (ICCCN)*, pages 1–9, July 2012.

[67] Cheng Yi, Alexander Afanasyev, Lan Wang, Beichuan Zhang, and Lixia Zhang. Adaptive forwarding in named data networking. *SIGCOMM Comput. Commun. Rev.*, 42(3) :62–67, June 2012.

[68] Yan ZHANG, Tao HUANG, Jiang LIU, Jian ya CHEN, and Yun jie LIU. Reverse-trace routing scheme in content centric networking. *The Journal of China Universities of Posts and Telecommunications*, 20(5) :22 – 29, 2013.

[69] Lorenzo Saino, Ioannis Psaras, and George Pavlou. Hash-routing schemes for information centric networking. In *ACM SIGCOMM*, ICN '13, 2013.

[70] W. V. Wollman and Y. Barsoum. Overview of open shortest path first, version 2 (ospf v2) routing in the tactical environment. In *Military Communications Conference, 1995. MILCOM '95, Conference Record, IEEE*, volume 3, pages 925–930 vol.3, Nov 1995.

[71] Lan Wang, AKM Mahmudul Hoque, Cheng Yi, Adam Alyyan, and Beichuan Zhang. OSPFN : An OSPF Based Routing Protocol for Named Data Networking. Technical report, Named Data Networking, 2012.

[72] A.K.M. Hoque, S.O. Amin, B. Alyyan, A.and Zhang, L. Zhang, and L. Wang. Nlsr : Named-data link state routing protocol. In *ACM SIGCOMM ICN WKSHPS 2013*, pages 15–20. ACM, 2013.

[73] Ali Marandi, Torsten Braun, Kavé Salamatian, and Nikolaos Thomos. BFR : a bloom filter-based routing approach for information-centric networks. *CoRR*, abs/1702.00340, 2017.

[74] J.J. Garcia-Luna-Aceves. Name-based content routing in information centric networks using distance information. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, ACM-ICN '14, pages 7–16, New York, NY, USA, 2014. ACM.

[75] Munyoung Lee, Kideok Cho, Kunwoo Park, Taekyoung Kwon, and Yanghee Choi. Scan : Scalable content routing for content-aware networking. In *ICC*, pages 1–5. IEEE, 2011.

[76] Vasilis Sourlas, Paris Flegkas, and Leandros Tassiulas. A novel cache aware routing scheme for information-centric networks. *Computer Networks*, 59 :44 – 61, 2014.

[77] João Vitor Torresa, Raouf Boutabab, and Otto Carlos M. B. Duartea. Evaluating cros-ndn : A comparative performance analysis of the controller-based routing scheme for named-data networking. 2016.

[78] O. Ascigil, V. Sourlas, I. Psaras, and G. Pavlou. Opportunistic off-path content discovery in information-centric networks. In *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pages 1–7, June 2016.

[79] E.J. Rosensweig and J. Kurose. Breadcrumbs : Efficient, best-effort content location in cache networks. In *IEEE INFOCOM 2009*, pages 2631–2635, 2009.

[80] Open Networking Foundation. Software-Defined Networking : The New Norm for Networks. White paper, Open Networking Foundation, Palo Alto, CA, USA, April 2012.

[81] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow : Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2) :69–74, March 2008.

[82] Damien Saucez and Thierry Turletti. Efficient caching in Content-Centric Networks using OpenFlow. *IEEE INFOCOM WKSHP 2013*, 2013.

[83] Xuan Nam Nguyen, Damien Saucez, and Thierry Turletti. Providing CCN functionalities over OpenFlow switches. Research report, INRIA, August 2013.

[84] Abhishek Chanda and Cedric Westphal. A content management layer for software-defined information centric networks. In *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, ICN '13, pages 47–48, New York, NY, USA, 2013. ACM.

[85] M. Vahlenkamp, F. Schneider, D. Kutscher, and J. Seedorf. Enabling information centric networking in ip networks using sdn. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–6, Nov 2013.

[86] Dukhyun Chang, Myungchul Kwak, Nakjung Choi, Ted Kwon, and Yanghee Choi. C-flow : An efficient content delivery framework with openflow. In *The International Conference on Information Networking 2014 (ICOIN2014)*, pages 270–275, Feb 2014.

[87] Dukhyun Chang, Junho Suh, Hyogi Jung, Ted "Taekyoung" Kwon, and Yanghee Choi. How to realize cdn interconnection (cdni) over openflow ? In *Proceedings of the 7th International Conference on Future Internet Technologies*, CFI '12, pages 29–30, New York, NY, USA, 2012. ACM.

[88] N. L. M. van Adrichem and F. A. Kuipers. Ndnflow : Software-defined named data networking. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, pages 1–5, April 2015.

[89] Luca Veltri, Giacomo Morabito, Stefano Salsano, Nicola Blefari-Melazzi, and Andrea Detti. Supporting information-centric functionality in software defined networks. *2012 IEEE ICC*, 2012.

[90] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, and L. Veltri. Information centric networking over sdn and openflow : Architectural aspects and experiments on the ofelia testbed. *Comput. Netw.*, 57(16) :3207–3221, November 2013.

[91] N. B. Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri. An openflow-based testbed for information centric networking. In *2012 Future Network Mobile Summit (FutureNetw)*, pages 1–9, July 2012.

[92] W. Liu, J. Ren, and J. Wang. A Unified Framework for Software-Defined Information-Centric Netowrk. Internet-Draft 1654, Internet Draft, August 2013.

[93] Abhishek Chanda, Cedric Westphal, and Dipankar Raychaudhuri. Content Based Traffic Engineering in Software Defined Information Centric Networks.

[94] Dimitris Syrivelis, George Parisis, Dirk Trossen, Paris Flegkas, Vasilis Sourlas, Thanasis Korakis, and Leandros Tassiulas. Pursuing a Software Defined Information-centric Network. *2012 European Workshop on Software Defined Networking*, pages 103–108, 2012.

[95] Mayutan Arumaithurai, Jiachen Chen, Edo Monticelli, Xiaoming Fu, and Kadangode K. Ramakrishnan. Exploiting icn for flexible management of software-defined networks. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, ACM-ICN '14, pages 107–116, New York, NY, USA, 2014. ACM.

[96] Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnSIM : NDN simulator for NS-3. Technical Report NDN-0005, NDN, October 2012.

[97] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnSIM 2.0 : A new version of the NDN simulator for NS-3. Technical Report NDN-0028, NDN, January 2015.

[98] D. Rossi and G. Rossini. On sizing ccn content stores by exploiting topological information. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 280–285, March 2012.

[99] Bruno Quoitin. Topologie abilene.

[100] GN3 Archive Project. Topologie geant. `https://geant3.archive.geant.org/Network/NetworkTopology/PublishingImages/topolgy_map.jpg`.

[101] Cesar Bernardini, Thomas Silverston, and Olivier Festor. MPC : Popularity-based caching strategy for content centric networks. *IEEE ICC 2013*, 2013.

[102] César Bernardini. *Stratégies de Cache basées sur la popularité pour Content Centric Networking*. PhD thesis, 2015. Thèse de doctorat dirigée par Festor, Olivier et Silverston, Thomas Informatique Université de Lorraine 2015.

[103] Named Data Networking Consortium. Ndn testbed. `http://named-data.net/ndn-testbed/`.

[104] Benjamin Rainer, Daniel Posch, Andreas Leibetseder, Sebastian Theuermann, and Hermann Hellwagner. A low-cost ndn testbed on banana pi routers. *Communications Magazine, IEEE*, 54(9) :6, oct 2016.

[105] Ze'ev Lailari, Hila Ben Abraham, Ben Aronberg, Jackie Hudepohl, Haowei Yuan, John DeHart, Jyoti Parwatikar, and Patrick Crowley. Experiments with the emulated ndn testbed in onl. In *Proceedings of the 2Nd International Conference on Information-Centric Networking*, ACM ICN 2015, pages 219–220, New York, NY, USA. ACM.

[106] Dirk Merkel. Docker : Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014.

[107] Named-Data Networking group. NDNx : The NDN Platform.

[108] Projet ANR DOCTOR. Deployment and securisation of new functionalities in virtualized networking environments. `http://www.doctor-project.org/`.

Résumé

L'Internet est un réseau de données mondial dont l'utilisation n'a cessé de croître ces dernières années. Popularisé auprès du grand public par l'utilisation d'applications telles que la messagerie électronique ou le Web au milieu des années 1990, l'Internet est devenu un vecteur essentiel de croissance économique dès les années 2000 avec l'apparition de nouveaux sites de commerce en ligne (Amazon, Ebay, Ali baba, etc.). La mise en place de nouvelles applications de distribution de contenus comme la vidéo à la demande ou en directe (streaming) (ex : Youtube) ou de partage de fichiers (P2P, etc.) a garanti son succès auprès d'un nombre d'utilisateurs toujours plus nombreux. De même, l'apparition de réseaux sociaux via des sites web permettant de s'échanger facilement du contenu entre communautés d'utilisateurs a accéléré davantage son développement (Facebook, Twitter, etc.). Ce nombre toujours croissant d'utilisateurs (environ 4 milliards) implique une quantité impressionnante de trafic transporté sur l'Internet et composé de contenu sous différentes formes (fichiers multimédia, streaming vidéo).

Toutefois, l'architecture de l'Internet, basée sur la suite protocolaire TCP/IP, n'a pas été conçue pour la distribution massive de contenus, tout comme elle n'a pas été créée pour des environnements mobiles et ubiquitaires, où chaque objet peut être connecté à l'Internet à tout instant et position, tout en se déplaçant. Ainsi, un nouveau type d'architecture pour un Internet du futur a vu le jour et se propose non plus d'être centré sur les hôtes de l'Internet comme l'architecture TCP/IP, mais de se focaliser sur le contenu lui-même. Ainsi, le contenu est adressé et non plus les hôtes, et le réseau est responsable de délivrer les données aux utilisateurs. Les nœuds du réseau peuvent conserver en mémoire les données qui transitent pour répondre aux futures requêtes, permettant de garder le contenu au plus proche des utilisateurs. Ces architectures orientées contenu (Information-Centric Networks) ont ainsi comme objectif de permettre de distribuer du contenu à large échelle et dépasser les limitations de l'Internet actuel pour s'adapter aux enjeux du moment. Parmi les architectures orientées contenu, l'architecture NDN (Named-Data Networking) a su agréger la plus importante communauté de chercheurs et est la plus aboutie pour un Internet du futur.

Dans le cadre de l'architecture NDN, au cours de ce doctorat, nous nous sommes concentrés sur les mécanismes de routage adaptés à cette nouvelle vision du réseau. En effet, la capacité à acheminer une requête vers la destination est fondamentale pour qu'une architecture réseau soit fonctionnelle et cette problématique avait été très peu étudiée jusqu'alors. Ainsi, dans ce manuscrit, nous proposons le protocole de routage SRSC (*SDN-based Routing Scheme for CCN/NDN*), qui repose sur l'utilisation du paradigme des réseaux logiciels (*Software-Defined Networks*, SDN). SRSC utilise un contrôleur capable de gérer le plan de contrôle du réseau NDN. En centralisant l'ensemble des informations telles que la topologie du réseau, la localisation des différents contenus et le contenu des mémoires cache des nœuds du réseau, le contrôleur va pouvoir établir la meilleure route pour acheminer les requêtes vers le contenu. SRSC permet également un routage de type *anycast*, c'est à dire qu'il permet d'acheminer les requêtes vers le nœud le plus proche qui dispose des données, permettant d'optimiser la distribution des requêtes dans le réseau et de

répartir la charge parmi tous les nœuds. De plus, SRSC utilise uniquement les messages *Interest* et *Data* de l'architecture NDN et tient son originalité du fait qu'il s'affranchit complètement de l'infrastructure TCP/IP existante.

Dans un premier temps, SRSC a été évalué via simulation avec le logiciel NS-3 où nous l'avons comparé à la méthode d'inondation des requêtes, appelée *flooding*, initialement proposée par NDN. SRSC a ensuite été implanté dans NDNx, l'implantation *open source* de l'architecture NDN, puis déployé sur notre testbed utilisant la technologie Docker. Ce testbed permet de virtualiser des nœuds NDN et d'observer un réel déploiement de cette architecture réseau à large échelle. Nous avons ainsi évalué les performances de notre protocole SRSC sur notre testbed virtualisé et nous l'avons comparé au protocole NLSR, (*Named-Data Link State Routing Protocol*), le protocole de routage du projet NDN.

Mots-clés : Réseaux orientés contenu, réseaux logiciel, routage, protocole, performance, Named-Data Networking, NDN, ICN, SDN