Architectures et fonctions avancées pour le déploiement progressif de réseaux orientés contenus

Xavier Marchal

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Architectures et fonctions avancées pour le déploiement progressif de réseaux orientés contenus

THÈSE

présentée et soutenue publiquement le 7 juin 2019

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Xavier Marchal

Composition du jury

Président :	Vincent Chevrier, Professeur des Universités (Hdr), Université de Lorraine
Rapporteurs :	Francine Krief, Professeur des Universités (Hdr), ENSEIRB Prométhée Spathis, Maître de Conférences (Hdr), UPMC
Examinateur :	Giovanna Carofiglio, Distinguished Engineer (Dr), Cisco Systems
Invité :	Damien Saucez, Chargé de Recherche (Dr), Inria
Encadrants :	Olivier Festor, Professeur des Universités (Hdr), Université de Lorraine Thibault Cholez, Maître de Conférences (Dr), Université de Lorraine

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 mon directeur de thèse, Olivier Festor, pour m'avoir accepté comme doctorant et pour son encadrement tout au long de cette thèse. Je remercie aussi grandement Thibault Cholez, co-directeur de cette thèse, pour son encadrement exemplaire, ses conseils ainsi que son aide dans les moments importants, mais aussi avec qui j'ai pu partager de très bons moments.

Je tiens également à remercier les membres du projet ANR DOCTOR avec qui j'ai pu travailler et plus particulièrement Bertrand Mathieu et Guillaume Doyen pour leur disponibilité et leur conduite du projet qui m'ont permis de travailler dans de bonnes conditions. Je tiens aussi à remercier Daishi Kondo et Tan Nguyen, deux doctorants avec qui j'ai pu travailler en plus étroite collaboration.

Je remercie Isabelle Chrisment, directrice de l'équipe projet Inria MADYNES (puis RESIST), pour m'avoir accueilli au sein de l'équipe ainsi que tous les autres membres de l'équipe qui ont pu répondre à mes questions et avec qui j'ai pu partager de bons moments tout au long de cette thèse.

Sommaire

Introduction générale

1 Contexte

Cette thèse a pour thématique l'évolution des réseaux informatiques et traite plus particulièrement du contexte technologique entourant le déploiement de nouveaux protocoles réseau, dits « orientés contenus », là ou un seul a présidé jusqu'ici et permis le développement de l'internet actuel : le protocole IP. Afin d'introduire la problématique, cette section présente tout d'abord le contexte de celle-ci, en rappelant dans un premier temps l'histoire du protocole IP, pour ensuite permettre de mieux comprendre la proposition des réseaux orientés contenu. Enfin, nous présenterons brièvement le cadre de travail particulier de ces travaux qui se sont inscrits dans un projet de recherche collaboratif national.

1.1 Petite histoire du protocole IP

L'ancêtre de l'Internet [1] actuel est le réseau ARPAnet [2], premier réseau informatique à commutation de paquet lancé aux états-unis en 1969. L'objectif était alors d'interconnecter les quelques ordinateurs de l'époque disponibles dans les centres de recherche et les universités américaines pour mutualiser les ressources de calcul, alors rares et coûteuses. La pile protocolaire d'Internet (TCP/IP) voit le jour en 1974 [CK74, VC] et permet de standardiser les communications sur ARPAnet. Le réseau ARPAnet (devenu NSFnet après scission du réseau militaire) croît rapidement mais reste cantonné au monde académique. La prochaine évolution majeure du réseau a lieu au début des années 90 avec l'avènement du web [3] qui en devient rapidement l'application principale et avec l'ouverture du réseau au trafic commercial. La démocratisation de l'informatique aidant, Internet a connu une forte croissance à la fin des années 90 et au début des années 2000. Les réseaux gagnant en capacité et les ordinateurs étant capables de manipuler des contenus multimédias, les pages web se sont progressivement complexifiées et enrichies, permettant l'avènement de toujours plus de services.

Dès lors, nous percevons l'écart entre l'usage initial qu'il était fait des communications TCP/IP, à savoir permettre l'exécution à distance de programmes de calculs sur quelques ordinateurs, avec l'usage qui en est fait aujourd'hui, à savoir la diffusion massive de contenus multimédias. Par ailleurs, d'importants ajouts ont dû être faits à la pile Internet pour couvrir des besoins non prévus initialement. Ainsi, la multiplication des ordinateurs connectés a nécessité un moyen de les identifier par un nom intelligible plutôt que par une adresse binaire (adresse IP de 32 bits), peu commode pour les humains, il s'agit du service DNS [4]. De même, le secret des communications a été permis a posteriori grâce aux protocoles de sécurisation des

1. https://fr.wikipedia.org/wiki/Internet
2. https://fr.wikipedia.org/wiki/ARPANET
3. https://fr.wikipedia.org/wiki/World_Wide_Web
4. https://fr.wikipedia.org/wiki/Domain_Name_System

communications par chiffrement, à savoir SSL puis TLS [5].

Il y a donc un problème d'ordre conceptuel entre l'architecture d'Internet qui est fondamentalement point à point (2 machines précisément identifiées par leur adresse communiquent entre elles) et son application principale qui est devenue la diffusion massive de contenus. En effet, les communications qui sont par nature bi-parties, comme peuvent l'être un appel téléphonique ou une connexion distante à un serveur représentent aujourd'hui une très faible part du trafic Internet par rapport à la diffusion de contenus. Les prévisions annuelles de l'entreprise Cisco [Cis18] prédisent ainsi que 81% du trafic Internet sera constitué par la diffusion de vidéos en 2021. Afin d'améliorer la diffusion de contenus sur Internet, des architectures spécialisées ont déjà été proposées avec succès au niveau applicatif dans les années 2000. Les réseaux orientés contenus en ont d'ailleurs judicieusement repris certains concepts pour les implanter au niveau du réseau lui-même.

1.2 L'émergence des réseaux orientés contenus

Pour répondre à ce besoin de diffusion de contenus à grande échelle, deux grandes familles de solutions ont été conçues au dessus des protocoles d'Internet, on parle de réseaux d'overlay, et essayent de dépasser le schéma point à point des communications pour gagner en efficacité : il s'agit des réseaux pair-à-pair (P2P networks) et des réseaux de diffusion de contenus (Content Delivery Networks ou CDN). Les premiers, développés au début des années 2000, sont la plupart du temps des réseaux ouverts qui permettent aux différents pairs les constituant de collaborer pour s'échanger des contenus. Un contenu peut ainsi être récupéré depuis plusieurs pairs en parallèle, ce qui en accélère le téléchargement. L'autre propriété intéressante est que les contenus les plus populaires seront naturellement répliqués chez davantage de pairs, augmentant ainsi en conséquence les ressources disponibles pour leur diffusion, ce qui forme ainsi un mécanisme de passage à l'échelle naturel. Les seconds sont des architectures réseau distribuées de serveurs proxy, opérées par des entreprises qui mettent ainsi à disposition de leurs clients, fournisseurs de contenus, de multiples serveurs localisés partout dans le monde de manière à pouvoir stocker les données populaires au plus proche des utilisateurs. L'architecture de routage interne du CDN permet ensuite d'utiliser le serveur le plus proche quand un utilisateur demande un contenu, ce processus étant transparent pour ce dernier. Ainsi, les utilisateurs de ces solutions ne savent pas a priori, c'est à dire au moment de leur requête, quelle machine en particulier, identifiée par son adresse IP, leur délivrera le contenu mais cherchent juste à recevoir le contenu lui-même, et de la manière la plus efficace possible.

La multiplication des utilisateurs et des services, toujours plus consommateurs de bande passante, exerce une forte pression sur les infrastructures réseaux sous-jacentes, alors même que les protocoles d'Internet sont reconnus pour ne pas être optimisés pour la diffusion de contenus. Devant ce constat, des chercheurs, notamment Van Jacobson [6] qui est l'inventeur du contrôle de congestion dans TCP, ont ouvert un nouveau domaine de recherche en réseau visant à proposer de nouveaux protocoles pouvant se substituer à TCP/IP en étant mieux adaptés aux enjeux actuels. Ces nouvelles architectures réseau sont dites orientées contenus (Information-Centric Networking ou ICN) et proposent un changement radical de paradigme où ce ne sont plus les machines mais les contenus qui sont adressables sur le réseau. Plusieurs innovations sont ainsi proposées. Tout d'abord, les contenus sont identifiés en tant que tels (et non plus à travers leur localisation) par un nom suivant un format d'URI (Uniform Resource Identifier), ce qui rend de

5. `https://fr.wikipedia.org/wiki/Transport_Layer_Security`

6. Google Tech Talks : A New Way to look at Networking, Van Jacobson, 2006. Available at `https://www.youtube.com/watch?v=gqGEMQveoqg`

fait caduque l'utilisation du service DNS. Ensuite, les contenus sont diffusés de manière multicast et potentiellement depuis plusieurs sources en parallèle, les requêtes concomitantes des utilisateurs étant mutualisées au niveau des routeurs qui en gardent la trace. Toujours dans l'optique d'améliorer la diffusion, les nœuds réseau disposent d'une capacité de mise en cache permettant de livrer plus rapidement les contenus populaires en évitant de solliciter le fournisseur originel. Enfin, la sécurité des communications bénéficie également de ce changement de paradigme. Ainsi, tout paquet transmis doit pouvoir être authentifié par la signature de son créateur. Cela permettrait de détecter et de révoquer des contenus malveillants. Par exemple, cela permettrait de lutter plus efficacement contre le spam car rien n'empêche actuellement un hôte de confiance, en l'occurrence un serveur mail contacté via une connexion sécurisée, de transmettre des contenus qui ne le sont pas. Les architectures ICN permettent donc, tout comme les CDN, de rapprocher les contenus des utilisateurs et, tout comme les réseaux P2P, d'avoir une diffusion multi-sources. Le fait d'implanter ces fonctionnalités directement au niveau réseau doit permettre de gagner en efficacité et en fonctionnalités.

1.3 Le projet ANR DOCTOR

Les travaux présentés dans ce manuscrit furent financés par un projet de l'Agence Nationale de la Recherche nommé « DOCTOR »[7] <ANR-14-CE28-0001> signifiant « DeplOyment and seCurisaTion of new functiOnalities in virtualized networking enviRonments ». Ce projet a réuni cinq partenaires, trois partenaires industriels, à savoir les entreprises Orange, Thales et Montimage, et deux partenaires académiques, à savoir l'Université de Technologie de Troyes et le LORIA, pendant quatre années, de septembre 2014 à septembre 2018. Le projet avait pour principale problématique la question suivante : comment permettre le déploiement et la sécurisation de nouveaux protocoles grâce à la virtualisation des fonctions réseau ? Plus particulièrement, le projet a pris comme cas d'étude le déploiement du protocole NDN et s'est attaché à fournir, outre la conception d'une architecture répondant aux besoins, l'implantation de composants logiciels prouvant la pertinence de l'approche proposée.

Mentionner le projet ANR DOCTOR est important pour permettre de bien contextualiser les travaux décrits dans ce manuscrit en termes d'originalité et de collaboration, et ce pour deux raisons. La première est que la vision portée par le projet DOCTOR se confond en grande partie avec celle portée par cette thèse, à savoir l'utilisation de fonctions réseaux virtualisées pour faciliter le déploiement et la gestion de réseaux orientés contenus. Par exemple, l'utilisation de passerelles de conversion protocolaire permettant de lier des îlots utilisant des technologies réseau différentes est une brique commune de la solution dans les deux cas. Ceci est parfaitement compréhensible dans la mesure où les productions de cette thèse devaient obligatoirement s'inscrire dans le programme scientifique du projet ANR DOCTOR. La seconde raison est que deux des quatre contributions principales ont été réalisées en collaboration avec d'autres membres du projet. Ainsi, le chapitre 4 a été réalisé en collaboration avec des collègues de l'UTT, et le chapitre 5 avec des collègues d'Orange et à nouveau de l'UTT.

2 Problématiques

Tout l'enjeu des travaux de recherche autour des ICN est de donner assez de garanties aux différents acteurs pour initier un mouvement progressif d'adoption de ces protocoles dans les réseaux. En effet, la pile protocolaire d'Internet est aujourd'hui implantée dans tout équipement in-

7. `http://www.doctor-project.org/`

formatique communicant et la remplacer demanderait un effort incommensurable. Celle-ci ayant peu voire pas évolué depuis le départ, on parle ainsi d'ossification d'Internet. Les innovations n'ont pu se faire qu'au niveau applicatif et par les machines terminales, les routeurs continuant d'opérer comme dans les années 80.

2.1 Défis majeurs

Une des grandes questions derrière la problématique générale de l'adoption des ICN est de savoir quelle force guidera le changement. Il y a en effet plusieurs acteurs en présence avec des besoins différents, à savoir les acteurs académiques, les fournisseurs d'accès à Internet, les fournisseurs de contenus et les clients finaux.

Les acteurs académiques, que ce soient les chercheurs ou les groupes de normalisation à l'IETF [8], ont d'ores et déjà largement fait évoluer les ICN durant la dernière décennie, si bien que les solutions les plus avancées comme l'architecture Named-Data Networking (NDN) sont aujourd'hui bien définies et fournissent des implantations de référence et peuvent donc être considérées comme des alternatives viables à la pile IP. Il reste bien entendu des aspects à améliorer comme nous le verrons dans l'état de l'art, mais l'état actuel permet déjà d'entrevoir des déploiements à court terme. Il est d'ailleurs intéressant de constater que de grands équipementiers réseau comme Cisco ou Huawei sont très actifs dans le domaine et pourraient donc potentiellement créer des routeurs adaptés rapidement.

Les fournisseurs de contenus et les clients ne seront probablement pas les forces motrices de ce changement. En effet, les solutions existant au dessus d'IP et évoquées précédemment (CDN, P2P) pallient pour l'instant efficacement ses faiblesses et permettent un niveau de qualité de service satisfaisant, bien que non optimal du point de vue du trafic réseau. Cependant des efforts doivent être faits pour amener les fournisseurs de contenus et les clients vers les premiers réseau ICN déployés car de tels réseaux sans contenu à véhiculer ni client à servir seraient sans objet. Un effort important doit donc être fait pour permettre une **interopérabilité** aisée avec les applications actuelles.

Les fournisseurs d'accès à Internet (FAI) semblent les plus à même d'initier le déploiement des ICN. Ils sont en effet les premiers impactés par l'augmentation du trafic qui nécessite en permanence d'importants et de coûteux d'investissements dans les infrastructures. Les ICN leur permettraient donc de réduire significativement le volume de trafic à acheminer, ce dernier étant davantage optimisé pour la diffusion de contenus. Cela leur permettrait également de monétiser l'utilisation des caches réseau, à l'instar des CDN actuellement. Cependant, les FAI ne peuvent migrer leur infrastructure réseau d'IP vers ICN sans un certain nombre de pré-requis. D'une part, les **performances** et la **sécurité** doivent atteindre un certain niveau de maturité afin de ne pas risquer d'impacter négativement la qualité de service offerte par le réseau une fois en production. D'autre part, les opérateurs ont besoin d'une solution facilitant le **déploiement et la configuration** du réseau ICN. En effet, celui-ci sera très certainement déployé de manière progressive à côté de leur infrastructure IP classique. Il faut donc qu'il puisse évoluer facilement de quelques nœuds initiaux vers un réseau complexe au fur et à mesure de sa croissance. De plus, un tel réseau doit permettre de répondre aux besoins opérationnels de supervision et de configuration des opérateurs, si possible en automatisant une grande partie de ces tâches.

8. `https://trac.ietf.org/trac/irtf/wiki/icnrg`

2.2 Problèmes traités

La problématique à laquelle cette thèse se propose d'apporter des réponses est la suivante : quels sont les principaux verrous opérationnels empêchant un déploiement progressif des réseaux orientés contenus et comment lever ceux-ci ?

Comme nous le verrons par l'étude de l'état de l'art, après quelques années de recherche, le paradigme ICN, et plus particulièrement l'architecture NDN, semblent assez mûrs pour permettre de premiers déploiements ambitieux. Paradoxalement, des questions importantes du point de vue opérationnel ne sont pas toujours bien traitées dans la littérature au profit de fonctionnalités additionnelles dont l'intérêt ne semble pas toujours évident. Cette thèse va traiter en particulier de quatre questions de première importance pour un opérateur.

La première concerne les performances, et plus particulièrement le débit atteignable par un fournisseur de contenus. Il est en effet important de pouvoir évaluer quel débit est atteignable par un réseau NDN et avec quelles ressources. Ou encore, quel est le composant limitant le débit sur le chemin d'une communication ? Il n'y a pas actuellement d'outil permettant de répondre facilement à ces questions par une approche expérimentale.

La seconde question concerne la sécurité de l'architecture NDN. De nouvelles attaques spécifiques à l'architecture ont été identifiées dans la littérature et permettent de réaliser des dénis de service sur les nœuds réseau. Elles ciblent en particulier les nouvelles structures internes des routeurs ICN comme la table servant à la mutualisation des requêtes ou encore le cache. Si le premier type d'attaque a été largement étudié, le second n'a pas encore été précisément caractérisé laissant planer un danger potentiel sur les premiers réseaux déployés.

La troisième question concerne l'interopérabilité entre un réseau NDN et les applications existantes. Comment permettre une interopérabilité aisée entre les sources actuelles de contenus, notamment le web, et un réseau NDN ? La difficulté est ici de concevoir un protocole d'adaptation qui respecte à la fois le protocole d'origine (sans en réduire les fonctionnalités) et le protocole NDN (sans en détourner le fonctionnement). De plus, au delà d'assurer l'interopérabilité, il faut également que les fonctionnalités propres au réseau NDN puissent pleinement s'exprimer (mutualisation des requêtes), ce qui est un défi étant donné le changement de paradigme.

La quatrième question concerne les modalités de déploiement et de gestion d'un nouveau réseau NDN. La question est ici de savoir quelle architecture offre le plus de facilités et de flexibilité pour pouvoir déployer progressivement un réseau NDN en parallèle du réseau IP. Les nouvelles architectures de virtualisation des fonctions réseau (NFV) offrent ici un cadre particulièrement intéressant mais n'ont pas été conçues pour déployer de services réseau sans partie IP. L'enjeu est ici de concevoir une telle architecture en créant puis en y intégrant des fonctions réseau spécifiques à NDN ainsi que les primitives de management associées.

3 Organisation des contributions

Les contributions de cette thèse s'organisent en trois grandes parties. La première présente l'état de l'art des travaux traitant des réseaux orientés contenus, et plus particulièrement de la résolution des différents problèmes et contraintes freinant leur adoption. La seconde partie décrit nos contributions visant à améliorer les performances et la sécurité de NDN. La troisième partie traite des solutions que nous proposons pour faciliter l'interopérabilité avec l'existant et le déploiement de NDN. Une conclusion générale termine ce manuscrit par un résumé et l'énumération des futurs travaux potentiels.

Première partie : État de l'art

Dans le chapitre 1, nous présentons les principes des réseaux orientés contenus et décrivons le fonctionnement de quatre architectures de tels réseaux. Nous détaillons plus particulièrement l'architecture Named-Data Networking qui sera choisie comme cas d'étude dans le reste du manuscrit. Enfin, nous présentons les principaux défis techniques que ces architectures doivent relever. Parmi les différents défis, le chapitre 2 dresse ensuite l'état de l'art des solutions aux principaux verrous opérationnels que nous avons identifiés, à savoir les problèmes de débit, de sécurité, d'interopérabilité et de déploiement. Pour chacune de ces problématiques, nous mettons en évidence une question majeure demeurant sans réponse satisfaisante à ce jour.

Deuxième partie : Performances et sécurité

Le chapitre 3 présente une première contribution s'intéressant aux faibles performances affectant les fournisseurs de contenus en temps réel souhaitant utiliser les ICN. Grâce à un outil de notre conception, ndnperf, et par une démarche expérimentale, nous mettons en évidence le coût processeur important de la génération de paquets NDN et proposons diverses optimisations permettant de le réduire grandement. Le chapitre 4 s'intéresse ensuite à un autre aspect critique, à savoir la sécurité du protocole NDN à travers l'étude de l'attaque par empoisonnement du cache. Cette attaque, bien que reconnue comme critique, reste en effet assez mal documentée et nous proposons dans un premier temps de la caractériser expérimentalement à travers différents scénarios. Nous mettons ainsi en évidence son efficacité et proposons enfin un correctif à l'implantation de référence du routeur NDN permettant d'empêcher le scénario d'attaque le plus efficace.

Troisième partie : Intéropérabilité et déploiement

Nous nous intéressons dans le chapitre 5 au problème d'interopérabilité de NDN avec les sources actuelles de contenus. Nous choisissons le web comme source première de contenus à bénéficier de NDN. Pour cela, nous concevons et évaluons une paire de *gateways* permettant de faire la traduction automatique HTTP/NDN grâce à un protocole d'adaptation. Les *gateways* forment ainsi un îlot NDN qui peut être utilisé de manière complètement transparente par les clients et les fournisseurs. Une attention particulière a de plus été portée pour faire bénéficier au mieux les contenus web du cache NDN. Pour terminer, le chapitre 6 propose une solution facilitant le déploiement et la gestion d'un réseau NDN grâce à une architecture orchestrée de microservices reprenant le principe de la virtualisation des fonctions réseau. Les différentes fonctionnalités d'un routeur sont ainsi décomposées en fonctions réseau virtualisées atomiques qui peuvent être composées au besoin grâce à un orchestrateur. Notre évaluation montre que ce dernier est capable de faire passer à l'échelle une fonction limitant le débit, ou encore de détecter une attaque et d'instancier les contre-mesures adéquates.

Première partie

État de l'art

Chapitre 1

Les réseaux orientés contenus

Sommaire

1.1 Introduction aux ICN

1.1.1 Les principes des ICN

Les réseaux orientés contenus, appelés Information-Centric Networking (ICN) en anglais, sont une nouvelle famille de protocoles réseau qui cherchent à améliorer les communications réseau étant donné l'usage actuel que l'on fait d'Internet. En effet, l'évolution des usages de l'Internet vers la diffusion massive de contenus (web, multimédia, etc.) et l'omniprésence de la connectivité ont demandé la mise en place de nouveaux éléments d'architecture comme la mise en place de CDN, la prise en charge de la mobilité, etc. De plus, de nombreuses propriétés essentielles aujourd'hui attendues d'un protocole réseau n'étaient pas offertes par la pile TCP/IP à l'origine, telles que la sécurité ou le fait d'avoir des adresses intelligibles. Par conséquent de nombreux ajouts à la pile TCP/IP sont aujourd'hui obligatoires tels que le service DNS ou le protocole de chiffrement TLS. Tous ces changements ont énormément complexifié la pile protocolaire réseau que l'on utilise car elle n'avait pas été pensée pour ces usages à l'origine. Ainsi les porteurs des réseaux ICN veulent faire table rase de cette suite protocolaire complexe et peu efficace en proposant des architectures réseau en accord avec l'utilisation que l'on a d'Internet aujourd'hui.

Ce nouveau paradigme est porté par Van Jacobson dont les premières publications sur ce sujet datent de 2006 [Jac06]. Les réseaux ICN reposent sur une approche différente de ceux que l'on utilise, majoritairement IP. Ces réseaux ont comme différence essentielle de mettre l'information

au centre des communications (on parle de réseaux information-centric ou encore content-centric) en lieu et place des machines (host-centric) et leurs connexions point à point. Les clients peuvent donc se concentrer sur l'information qu'ils recherchent sans se soucier de sa localisation exacte sur le réseau. Ainsi on ne cherche plus à communiquer avec une entité en particulier pour récupérer un contenu car n'importe quel nœud du réseau peut potentiellement fournir le contenu demandé. Les réseaux ICN utilisent en effet des caches locaux qui peuvent être gérés par les nœuds du réseau (on-path) ou une autre entité (off-path), à l'instar des CDN, pour optimiser la distribution de contenus. Mais cela ne se fait pas n'importe comment, les noms utilisés pour un contenu sont supposés uniques et le contenu associé à un nom est supposé immuable pour pouvoir garder une certaine cohérence. De plus, les contenus sont signés pour pouvoir être authentifiés par le client au niveau du protocole (donc pas de surcouche) et c'est cette propriété qui permet la réutilisation des paquets pour répondre ultérieurement à des requêtes similaires, et ce sans qu'il ait besoin de faire confiance au nœud émetteur.

La suite de cette section présente les principales architectures ICN [XVS+14, ADI+12], avec un accent mis sur l'architecture NDN qui est privilégiée dans cette thèse, puis nous conclurons sur les principaux défis de ces réseaux.

1.1.2 Data Oriented Network Architecture (DONA)

DONA [KCC+07] est une architecture de réseau orienté contenus initiée en 2006 et l'une des premières architectures ICN à avoir vu le jour. DONA a été conçue par l'Université de Californie - Berkeley, et reprend selon ses auteurs des éléments de design de TRIAD (Translating Relaying Internet Architecture integrating Active Directories), HIP (Host Identity Protocol) et SFS (Self-certifying File System).

DONA fonctionne toujours au dessus de IP, mais revoie la façon dont la découverte de contenus est réalisée en répondant à plusieurs problématiques comme la persistance, la disponibilité et l'authentification des données. Ainsi *DONA* se place en concurrent à DNS (Domain Name System) en utilisant un système de résolution de noms qui suit un plan de nommage plat et auto-certifié.

Les noms utilisés dans *DONA* ont deux composants, P et L. Le premier composant est le résultat d'une fonction de hachage de la clé publique du serveur proposant le contenu, le second composant est un label défini par le serveur pour identifier son contenu. Ainsi, un contenu peut être identifié avec un nom de la forme $P{:}L$ (difficilement lisible par un humain) défini comme indépendant de l'application, de la localisation et globalement unique.

Un réseau *DONA* est constitué de nœuds appelés *resolution handlers* (RH). Pour résoudre des noms, le réseau se base sur 2 types de messages, *FIND* et *REGISTER*. Le premier est utilisé par les utilisateurs pour localiser un contenu et les nœuds RH seront en charge de router ce message vers une copie proche. Pour les messages *REGISTER*, un serveur va envoyer à un nœud RH autant de messages qu'il a de contenus à diffuser. Ce nœud RH sera en charge de propager le message du serveur vers les autres nœuds RH. Chaque nœud maintient une table des contenus qui fait correspondre un nom à un *next-hop* ainsi que sa distance vers la copie.

Pour la résolution de nom, les nœuds RH utilisent une résolution par plus long préfixe. Cela se traduit par le fait que s'il n'existe pas de route pour un nom $P{:}L$, la requête pourra toujours être routé vers $P{:}*$. Si il n'existe pas d'entrée, un nœud RH peut envoyer la requête à son parent. Quand une copie est trouvée, elle peut être, au choix, envoyée par le chemin inverse ou directement à l'utilisateur. Lors de la réception d'une réponse, l'utilisateur reçoit au minimum le triplé (*data*, *public key*, *signature*) pour pouvoir vérifier le contenu mais d'autres méta-info peuvent être associées.

Les nœuds RH peuvent fournir des fonctions supplémentaires avec par exemple une fonction de cache. Ainsi un nœud RH va changer les adresses IP des requêtes *FIND* pour pouvoir mettre en cache leur réponses et pouvoir les redistribuer à son tour.

Ainsi *DONA* propose un mécanisme de résolution de noms qui pourrait remplacer le protocole DNS que nous utilisons aujourd'hui, mais de manière efficace et sans avoir besoin d'un service tiers pour la gestion de certificats. *DONA* utilise toujours IP comme protocole de transport, sa mise en place reste donc relativement simple puisque qu'elle ne fait qu'ajouter un service.

1.1.3 Network of Information (NetInf)

Basée sur l'architecture SAIL (Scalable and Adaptive Internet Solution), NetInf peut fonctionner de manière hybride. Il utilise un nommage dont les noms sont sous la forme d'URI « ni ://A/L » et sont composés de deux parties : la première, A, correspond à l'autorité ayant émis le contenu et la deuxième, L, permet de décrire le contenu. Ces deux parties peuvent aussi bien être des hashs que des structures plus hiérarchiques comme des URL. Selon la façon de router les noms, le nommage peut être considéré comme plat lors de la comparaison de noms ou hiérarchique pour le routage qui utilise une résolution par plus long préfixe. Le routage peut être réalisé de façon couplée ou découplée.

Dans le premier cas, un *Name Resolution System* (NRS) est utilisé pour faire correspondre un nom à une localisation. Le NRS est une sorte de table de hachage distribuée (DHT) à plusieurs niveaux où la résolution de la partie L se fait au niveau local, alors que la résolution de la partie A se fait au niveau global. Pour ce faire, un serveur va envoyer des messages *PUBLISH* avec leur localisateur au NRS local, qui a son tour va concaténer toutes les parties L pour une même autorité A dans un filtre de Bloom et va l'envoyer via un message *PUBLISH* vers le NRS global. Ainsi, quand un utilisateur est intéressé par un contenu, il demande à son NRS local qui peut consulter à son tour le NRS global pour pouvoir retourner un localisateur que le client pourra utiliser pour récupérer le contenu. En envoyant un message *GET* au serveur, le client recevra un message *DATA* en retour avec le contenu souhaité.

Dans le deuxième cas, le réseau utilise un protocole de routage pour peupler les tables de nœuds appelés *Content Routers* (CR). Lorsqu'un message *GET* est reçu par un CR, celui-ci le propage de proche en proche. De même que pour le premier cas, un message *DATA* est retourner à l'utilisateur en suivant le chemin inverse, ce qui est possible grâce à l'ajout d'informations de routage dans le message *GET* au fur et a mesure de sa progression dans le réseau. Les CR peuvent aussi mettre en cache les contenus qui les traversent. Il est aussi possible de déployer des caches locaux, qui seront considérés comme des serveurs (similaires aux CDN) et ceux-ci peuvent être organisés en arbre où les caches locaux seraient de petite taille et les autres de plus en plus gros au fur et a mesure que l'on se rapproche de la racine.

Il est aussi possible de mixer ces deux méthodes, les NRS vont ainsi donner des suggestions de routage pour transmettre la demande dans un voisinage proche pour que des CR puissent continuer le routage jusqu'au serveur. Cela peut aussi fonctionner dans l'autre sens quand un CR n'a pas assez d'information de routage il peut demander des « indices » vers où transmettre la requête pour que le routage puisse continuer.

1.1.4 Publish-Subscribe Internet Technologies (PURSUIT)

PURSUIT est un projet dans la continuité de PSIRP (Publish-Subscribe Interest Routing Paradigm) et a été proposé dans le cadre du programme de recherche européen FP7. Cette architecture peut complètement remplacer IP avec un modèle basé sur la publication/souscrip-

tion. L'architecture est basée sur trois grandes fonctions : le rendez-vous, le management de la topologie et la transmission des données. L'ordre de fonctionnement est tel que la fonction de rendez-vous va faire correspondre les souscriptions aux publications, qui seront ensuite utilisées par la fonction de management pour établir les routes entre les deux parties, puis pour finir utiliser ces routes pour la transmission des données.

PURSUIT utilise un nommage plat composé de deux composants. Le premier, nommé *scope ID*, regroupe des informations relatives au contenu. Il est possible qu'un même contenu ait plusieurs *scope ID*. Le second, nommé *rendezvous ID*, représente l'identité du contenu sur le réseau.

La résolution des noms se fait au niveau des nœuds de rendez-vous. Ceux-ci fonctionnent comme une DHT hiérarchique. Un serveur va envoyer des messages *PUBLISH*, pour enregistrer ses contenus au nœud de rendez-vous local, qui se chargera de les transmettre à celui qui gère le *scope ID* de ces contenus. Lorsque qu'un utilisateur voudra demander un contenu, il enverra un message *SUBSCRIBE* à son nœud de rendez-vous local qui se chargera de transmettre la requête jusqu'au bon nœud. Si il y a une correspondance, le nœud de rendez-vous qui gère le *scope ID* du contenu demandé va contacter le manager de topologie pour créer une route entre le serveur et l'utilisateur pour le transfert du contenu. Pour effectuer le routage, le manager de topologie va calculer un filtre de Bloom qui sera intégré au paquet. Ainsi les routeurs n'auront qu'a effectuer une opération simple pour savoir où l'envoyer.

En terme de cache, PURSUIT est plus enclin à utiliser des nœuds de cache off-path à cause de la création de routes dynamiques, rendant peu efficace la mise en place de cache on-path. Ces nœuds vont jouer un rôle très similaire aux CDN que nous utilisons, il vont agir comme des serveurs en enregistrant les contenus qu'ils cachent aux nœuds de rendez-vous et les distribuer de la même façon que le serveur d'origine.

1.2 Named Data Networking (NDN)

1.2.1 Principes généraux

NDN [JST+09] [ZAB+14] est l'une des propositions de protocoles ICN les plus populaires, et est par conséquent le candidat le plus prometteur pour un déploiement potentiel. Certaines propriétés de NDN décrites dans cette section sont partagées avec l'architecture Content-Centric Networking (CCN). En effet NDN et CCN sont très proches, le projet CCN a débuté en 2007 au Palo Alto Research Center (PARC) mais ses principes étaient déjà présentés durant un *talk* de Van Jacobson à Google en 2006. CCN a depuis été racheté par Cisco. NDN, quant à lui, est un *fork* de CCN effectué en 2009 à cause d'une divergence d'opinions sur l'efficacité de l'encodage des paquets. L'architecture NDN est open-source contrairement à CCN qui est restée propriétaire. Nous présentons ici l'ancien format de paquet de NDN (0.2) pour rester cohérent avec nos travaux, mais les différences avec le nouveau format de paquet (0.3) seront abordées en même temps que celles avec CCN.

NDN [Bra09, NDN] se place au niveau de la couche 3 (réseau) et 4 (transport) du modèle OSI, car la notion d'*host* et donc de connexion n'existe plus. Il est donc possible de déployer NDN directement sur Ethernet mais aussi en overlay d'IP pour faciliter la mise en place des premiers réseaux. NDN repose sur un réseau basé sur des contenus nommés et organisés selon un plan de nommage hiérarchique similaire à celui des URI. Les noms NDN servent aussi bien à router les paquets NDN, qu'à décrire un contenu. Ainsi un serveur va enregistrer un préfixe NDN sur un routeur pour établir une route et spécifiera le contenu en tant que suffixe. Ces deux parties forment un nom NDN (voir Figure 1.1). Pour transmettre des données, le protocole NDN

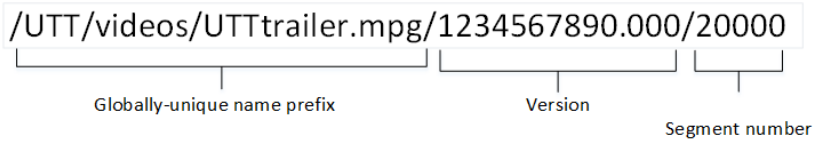


FIGURE 1.1 – Exemple de nom NDN

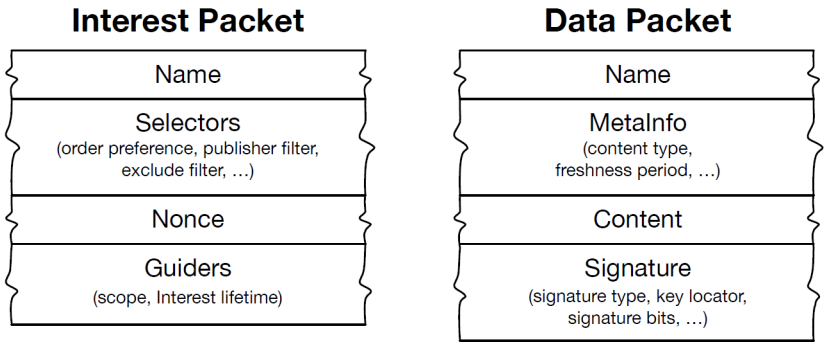


FIGURE 1.2 – spécification des paquets NDN [NDN]

se base sur deux types de paquets (Figure 1.2) : les paquets *Interest* et les paquets *Data*. Ces paquets peuvent avoir une taille maximale de 8800 octets et sont encodés selon le format TLV (Type-Length-Value). Ainsi un paquet NDN n'a pas de forme définitive et pourra évoluer dans le temps.

Les paquets *Interest*, qui permettent d'effectuer une requête, doivent être au minimum composés d'un nom NDN et d'un nonce. Le nonce est un entier de 32 bits qui permet pour un nom donné d'avoir un paquet *Interest* théoriquement unique. Le principal intérêt de cette propriété est d'éviter aux paquets *Interest* d'effectuer des boucles dans le réseau. Il existe aussi, en plus du nom et de la nonce, deux types de champs supplémentaires optionnels : les sélecteurs et les guides. Les sélecteurs permettent de préciser la requête, il en existe six : *MinSuffixComponents*, *MaxSuffixComponents*, *PublisherPublicKeyLocator*, *Exclude*, *ChildSelector* et *MustBeFresh*. Les deux premiers servent à définir le nombre minimum et maximum de *NameComponents* qui doivent suivre le nom utilisé dans le paquets *Interest*. Le troisième permet de spécifier le nom de la clé qui devra être utilisée pour signer le paquet *Data* résultant, ce qui permet d'une certaine façon de « selectionner » le fournisseur de contenus. Le quatrième permet d'exclure une liste de *NameComponents* qui pourrait suivre le nom du paquet envoyé, ce qui permet d'explorer ou d'éviter certains contenus en particulier ceux partageant un même préfixe. Le cinquième permet de spécifier, si on le veut, le membre le plus à gauche ou le plus à droite, cela pouvant par exemple, se traduire dans le cas ou le *NameComponent* suivant serait un numéro de version comme la plus vieille ou la plus récente d'un contenu. Le dernier précise dans le cas où la réponse à cette requête a déjà été envoyée et est encore présente dans le cache des routeurs si l'on accepte un contenu expiré ou non. Les guides servent à spécifier le comportement des routeurs avec ces paquets. Il n'y a actuellement qu'un seul guide nommé *InterestLifeTime* et qui spécifie combien de temps le routeur va garder l'entrée de ce paquet avant de la supprimer.

Pour répondre aux paquets *Interest*, le protocole NDN prévoit un autre type de paquet nommé *Data*. Celui-ci reprend le même champ de nom NDN pour permettre le routage inverse du contenu au client, les autres champs lui étant propres. Le champ *MetaInfo* regroupe actuellement trois

variables permettant de décrire le contenu : *ContentType* qui décrit le type de contenus avec quatre valeurs possibles (BLOB, LINK, KEY, NACK), *FreshnessPeriod* qui indique combien de temps, en millisecondes, la donnée et considérée comme valide après réception et *FinalBlockId* qui, lorsque le contenu ne peut pas être envoyé en un seul paquet, permet aux utilisateurs de savoir quel est le dernier segment du contenu (le numéro de segment est ajouté à la fin du nom du paquet). Le champ *Content* contient les données du contenu et le champ *Signature* contient la signature qui peut être pour le moment soit RSA(-2048 par défaut), ECDSA ou un simple hash SHA-256, ainsi que les informations associées à cette signature comme le nom de la clé publique utilisée, etc...

Il est possible d'associer un protocole de lien aux paquets NDN, nommé NDNLPv2 (Named Data Networking Link Protocol). Il prend la forme d'un entête qui s'ajoute avant le paquet NDN pour ajouter des informations relatives aux transferts des paquets. Ce protocole utilise toujours le format TLV pour l'encodage et permet notamment de gérer plus efficacement la fragmentation et la gestion des flux avec un mécanisme d'acquittement et d'informations relatives à l'état des liens, ou encore donner ou recevoir des informations du nœud local.

1.2.2 Fonctionnement d'un nœud NDN

Pour router les noms NDN, les développeurs ont mis à disposition un programme sous forme de *daemon*, nommé NFD (NDN Forwarding Daemon), qui va se charger du routage. Celui-ci est constitué de 3 tables primordiales pour le fonctionnement de NDN (voir Figure 1.3) :

— Le *Content Store* (CS) : c'est dans cette table que toutes les réponses qui ont pu satisfaire une requête sont stockées. Elle peut être gérée à l'aide de différentes stratégies de cache comme par exemple Least Recently Used (LRU) ;

— La *Pending Interest Table* (PIT) : c'est dans cette table que sont regroupées les informations provenant des requêtes en attente d'une réponse. Il existe une entrée pour chaque nom de contenu en attente de sorte que si deux requêtes pour un même contenu sont reçues, NFD va enregistrer dans une liste toutes les *Faces* (les *Faces* sont une généralisation des interfaces pour les réseau IP) par lesquelles ces requêtes sont arrivées pour pouvoir renvoyer par la suite la réponse par ces même *Faces* et ainsi satisfaire tous les clients concomitamment ;

— La *Forwarding Information Base* (FIB) : c'est dans cette table que NFD stocke les différentes routes disponibles. La FIB utilise comme clé des préfixes NDN qui peuvent être créés de 3 manières :

1. enregistrés par un serveur via un paquet *Interest* signé sur un préfixe spécifique ;
2. Manuellement par l'administrateur du routeur ;
3. Transmis via le protocole de routage NDN, NLRS (*Named Data Link State Routing Protocol*).

et comme valeur la liste des *Faces* disponibles pour ce préfixe. Les préfixes peuvent être agrégés, à l'instar des sous-réseaux consécutifs dans le monde IP, pour réduire la taille de la table. NFD gère chaque préfixe indépendamment grâce à une stratégie de routage qui lui permettra de choisir parmi les différentes *Faces* disponibles vers celle ou celles vers lesquelles celui-ci va transmettre le paquet *Interest* (*best-route*, *load-balancing*, *multicast*, etc...).

Chacune de ces tables joue un rôle précis durant le processus de routage (voir Figure 1.4) :

1. Lors de la réception d'une requête (*Interest*), NFD va d'abord regarder dans le CS pour voir si il n'y a pas déjà eu une réponse à cette requête. Si c'est le cas, alors il renvoie la

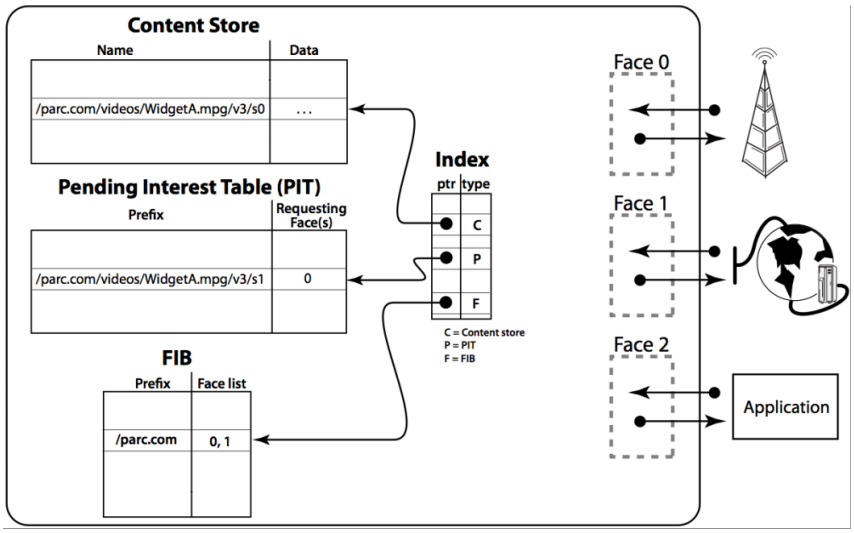


FIGURE 1.3 – Structures de données dans NFD servant au routage NDN [NDN]

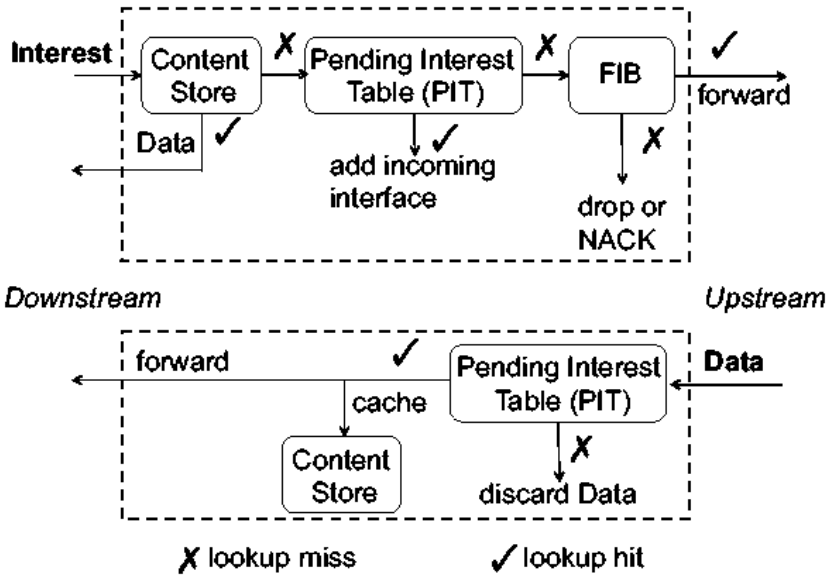


FIGURE 1.4 – Étapes de traitement des flux des paquets *Interest* et *Data* dans NFD [Pro]

donnée depuis le cache, sinon il continue le processus de routage ;

2. NFD va alors enregistrer le préfixe de la requête dans la PIT puis,

3. NFD va regarder dans la FIB si une entrée existe pour ce préfixe. Si c'est le cas alors, il va transmettre la requête au(x) serveur(s) ou au(x) prochain(s) routeur(s) ;

4. Après être arrivé au serveur, celui-ci forge une réponse (*Data*) et l'envoie à un nœud NFD ;

5. NFD va ensuite vérifier si le préfixe de la réponse correspond à une entrée de la PIT. Si au moins une entrée correspond, alors NFD mettra en cache la réponse puis la transmettra à toutes les *Faces* par où sont venues les requêtes.

En observant le trafic dans les deux sens, un nœud NDN peut garder des statistiques sur les différents chemins avec une granularité au niveau des préfixes et ils peuvent adapter leur stratégie de routage par préfixe pour éviter la congestion ou des erreurs de liens sans avoir à solliciter l'utilisateur.

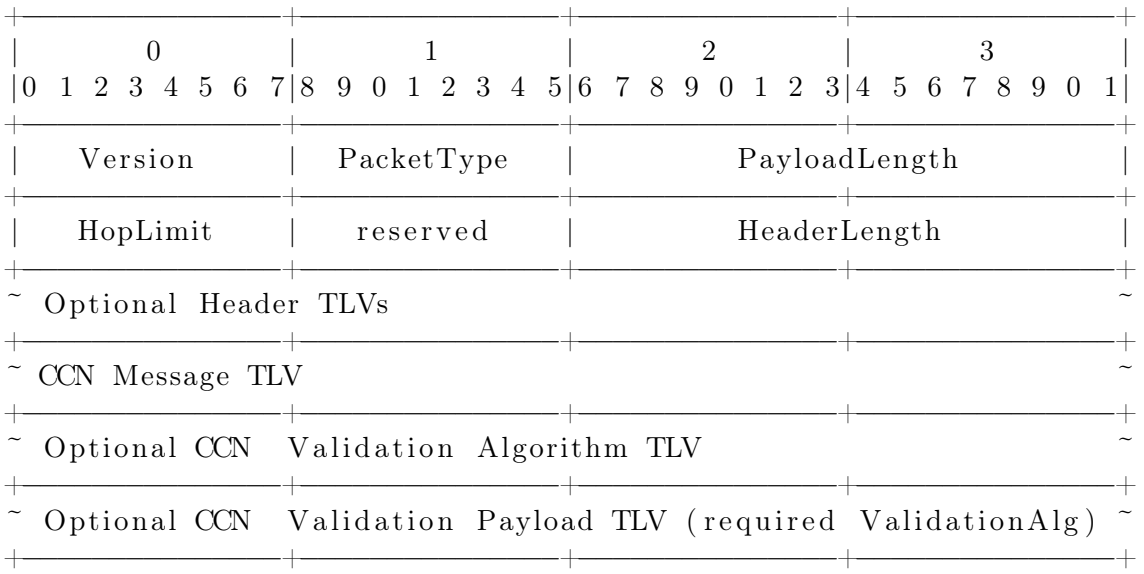
Dans les faits, l'implémentation de NFD est un peu différente pour le routage des paquets *Interest*. La PIT est consultée avant le *Content Store*, pour optimiser les accès au CS pour les contenus populaires.

1.2.3 Divergences et convergences entre NDN et CCN

CCN suit actuellement les spécifications CCNx 1.0 [MSUW15, Sem]. Le format de paquet de CCN est légèrement différent de NDN ainsi que la façon dont ceux-ci sont routés. Pour le format de paquet, il est défini dans le Listing 1.1. Un paquet est constitué de deux entêtes, l'un obligatoire et l'autre optionnel. Pour le premier entête, nous avons un champ pour indiquer la version du protocole, le type de paquet qui est transporté, ainsi que sa taille (nombre d'octets après l'entête optionnel), le nombre maximum de sauts (similaire à IP) et la taille de l'entête optionnel. Dans CCN, il est nécessaire de spécifier un nombre maximum de saut car il n'existe pas d'autre système pour empêcher les boucles contrairement à NDN où ce problème est résolu par le nonce inclus dans le paquet *Interest*. Le second entête est une structure TLV ; Il contient des informations supplémentaires qui n'ont pas de rapport direct avec la donnée. On peut retrouver par exemple des informations sur la fragmentation du paquet. Cet entête peut être comparé à NDNLP.

Nous avons ensuite le cœur du paquet, celui-ci pourra être défini comme un *Interest*, un *Content Object* (similaire au *Data*) ou un *Interest Return* (pour les NACK lorsqu'un routeur ne connaît pas de route pour l'*Interest*) avec un champ spécifique ainsi que la taille du message. Il en va de même pour définir le nom, avec un champ pour l'identifier et sa taille. Pour finir, deux champs TLV optionnels peuvent être définis pour les méta-datas suivis d'un *payload* pour les deux types de paquets, ce qui n'est possible que pour les *Data* dans la version de NDN que nous avons décrite précédemment. Le paquet se termine avec des champs pour contenir une éventuelle signature et fonctionne de manière similaire à NDN.

Listing 1.1 – Format de paquet pour CCNx 1.0 [MSUW15]



Concernant l'encodage du paquet, la gestion des champs définissant les tailles sont de taille fixe, défini à 2 octets, contrairement à NDN où ils sont flexibles avec un système à 1-3-5-9 octets. Ce choix, même si il est moins efficace du point de vue de l'encodage, est justifié par un traitement plus rapide des paquets. Pour le routage des paquets dans CCN, les correspondances sont uniquement strictes, ainsi il n'est pas possible de répondre avec un nom plus long que celui demandé et le paquet *Content Object* doit aussi valider des restrictions imposées par le paquet *Interest* (*KeyIdRestr* et *ContentObjectHashRestr*) si elles sont présentes. Cela permet encore une fois de traiter le paquet plus rapidement car pour valider une décision, il suffit de comparer les noms comme des chaînes binaires. NDN quant à lui permet une décision par préfixe, ce qui nécessite un traitement plus lourd car chaque composant du nom doit être traité. Une autre différence est qu'il n'est pas possible de récupérer des paquets périmés en cache dans CCN contrairement à NDN où c'est un choix laissé à l'utilisateur (champ *MustBeFresh*). De plus, pour CCN, les temps définis dans les paquets sont majoritairement des valeurs absolues, seul celui équivalemment à *InterestLifeTime* est relatif.

La spécification des paquets NDN a aussi changé durant nos recherches. Parmi ces changements, on peut noter l'apparition d'un champ *ForwardingHint* et de paquets *Data* « spécialisés » qui permettent de donner et d'utiliser des chemins alternatifs pour récupérer un contenu, un peu à la manière des redirections de HTTP. Avec la révision 0.3 du format de paquet, d'autres champs ont été ajoutés pour les paquets *Interest* avec *CanBePrefix* pour définir si ce paquet peut être répondu par un paquet *Data* dont le nom du premier est préfixe du deuxième, *HopLimit* pour définir le nombre maximum de sauts et *Parameters* pour transporter des données dans le paquet (nécessite d'ajouter le hash de la valeur du champs dans le nom). Par contre, la quasi totalité des champs de type *Selector* disparaissent. Ainsi les champs suivants deviennent obsolètes : *MinSuffixComponents*, *MaxSuffixComponents*, *PublisherPublicKeyLocator*, *Exclude* et *ChildSelector*. Seul *MustBeFresh* est gardé. Ces changements ne remettent pas en causes nos travaux même s'ils peuvent modifier la façon dont les contenus sont demandés lors d'attaques comme nous le verrons dans le chapitre 4, ou faciliter le transfert de données dans certains cas abordés dans le chapitre 5.

On peut remarquer que les deux derniers champs ajoutés sont très similaires à ceux de

la spécification CCNx 1.0. Cela vient très certainement de l'effort d'harmonisation des deux spécifications initiée en 2017 par les deux communautés au sein du groupe de travail d'ICNRG. Ainsi le draft [CfNoICN] dresse les différences apportées aux deux spécifications depuis le *fork*.

Les deux spécifications NDN et CCNx partagent donc une base historique commune. Même si elles sont différentes, elles gardent des objectifs très similaires. Cela est d'autant plus marqué par le récent effort d'harmonisation initié par les deux communautés. Cette harmonisation est bienvenue pour créer un standard à même d'attirer l'attention d'industriels souhaitant tester leur déploiement. On peut remarquer que les spécifications de CCNx semblent plus orientées sur l'efficacité du protocole alors que celles de NDN semblet plus orientées sur la flexibilité. Ainsi, puisqu'il fallait choisir une architecture en particulier dans le cadre de nos travaux, l'architecture NDN nous a semblé plus adaptée pour de la recherche académique car issue d'un projet public (NSF), et maintenue par une université et dont l'ensemble des développements est open source.

1.3 Les principaux défis des ICN

La RFC 7927 [Cha] dresse une liste importante des défis scientifiques que les architectures ICN doivent relever pour être viables. La majorité est liée à la sécurité des protocoles ICN, le routage des paquets et leur stockage dans le réseau.

En terme de sécurité, le point le plus important est comment authentifier avec certitude qu'un contenu a bien été généré par la bonne entité et qu'il n'a pas été modifié en cours de route. Puisque le concept de connexion n'existe plus, nous ne pouvons plus nous en remettre à des protocoles comme TLS et le contenu peut provenir de n'importe où car il sera très probablement répliqué par des caches. Les réseaux ICN utilise majoritairement deux types de nommage : plat ou hiérarchique. Le premier nommage a la particularité de pouvoir inclure dans le nom du contenu une partie cryptographique, ce qui lui donne la propriété d'être auto-certifié. Ainsi, il n'est pas nécessaire dans ce cas de dépendre d'une PKI (*Public Key Infrastructure*) pour vérifier un contenu. Par contre les noms associés aux contenus risquent d'être impossibles à comprendre pour un humain car composés de chaînes binaires. Pour le deuxième nommage, les noms ressemblent plus à des URI, bien plus simples à comprendre pour un humain, qui offrent un routage plus efficace car cela permet d'agréger les routes et de faire du routage par préfixe. Ainsi, pour le premier, le problème concerne majoritairement le routage et la découverte des noms et pour le second, la gestion de l'authentification des contenus. Pour finir, dans les deux cas, il semble se poser la question de lier une identité au « monde réel » comme nous le faisons aujourd'hui avec les certificats. Malheureusement, il ne semble pas y avoir encore de solution efficace pour répondre à la question de la gestion de la chaîne de confiance pour les réseaux ICN.

Le document pose aussi des problématiques liées au routage et à la gestion des nœuds du réseau. Ainsi il semble que les différentes architectures ICN n'utilisent que très peu des stratégies de broadcast mais se basent sur un routage de proche en proche ayant pour conséquence que les nœuds du réseau utilisent rarement les chemins optimaux à cause d'une vision restreinte du réseau et en l'absence d'un protocole de routage efficace. Il se pose aussi la question de la gestion du contrôle de la congestion, car contrairement à TCP/IP où celle-ci est faite par les *endpoints*, il n'est pas possible de faire de même dans une architecture ICN et se sont aux nœuds du réseau de réaliser cette gestion. Se pose alors le problème d'allouer la bande passante de manière équitable lorsqu'il est difficile d'identifier les utilisateurs et leurs flux de données. En terme de gestion de ressources, il est noté une gestion des caches peu efficace notamment à cause du manque de contexte car trop bas niveau. Les nœuds ont aussi tendance à consommer beaucoup de ressources pour le routage (CPU, RAM) et cela les rend d'autant plus sensibles à des attaques par déni de

services comme la pollution de cache, l'inondation de requêtes pour saturer les tables du nœud, etc. Comme pour le problème de la congestion, la difficulté d'identifier les flux de données rend difficile la protection contre certaines de ces attaques.

La gestion des droits d'accès semble difficile dans un réseau ICN. Un utilisateur ne communiquant pas directement avec un fournisseur de contenus, il est ainsi compliqué pour celui-ci de savoir avec qui il communique et la présence de cache n'arrange pas les choses car il devient difficile de contrôler la distribution de ses contenus. Des approches imparfaites semblent avoir été proposées comme la mise en place d'un service tiers non-ICN pour la gestion des droits ou l'usage du chiffrement avec une gestion temporelle de clés assez peu pratique (*broadcast encryption*). Cela pose donc les problèmes du chiffrement et de la vie privée sur les réseaux ICN, car sans un système de gestion de droits, il est difficile de chiffrer des données entre deux parties et donc de proposer un vrai anonymat. Pour la vie privée un opérateur peut donc savoir se qu'un utilisateur demande (comme pour HTTP) mais aussi les autres utilisateurs proches car demander un contenu laisse des traces plus ou moins longue dans les caches des nœuds du réseau et, en analysant ces caches périodiquement, il est possible de savoir ce que les utilisateurs demandent via ces nœuds. Les architectures ICN étant pensées pour la diffusion de masse, le cas de la gestion de droits semble être un faux-problème car la logique même des transferts de données tend vers l'ouverture alors que celle-ci cherche à faire le contraire, autant utiliser des protocoles comme IP, qui sont plus adaptés à cette problématique.

Un des challenges les plus importants qui devra être résolu est la réalisation de communications de type *PUSH*. De par leur nature, les communications dans les architectures ICN sont exclusivement gérées par le client (*PULL*). Cela fonctionne bien pour certains services comme la récupération de contenus web, le streaming, etc, mais beaucoup d'autres ont besoin du modèle *PUSH* notamment pour les applications temps réel comme les websockets ou encore les applications IoT. Des propositions ont déjà été publiées mais ce challenge reste un sujet majeur dans la communauté où il était d'ailleurs traité lors de la dernière conférence ACM ICN'18.

1.4 Conclusion

Nous venons de présenter quatre architectures ICN parmi les plus importantes. Nous avons choisi l'architecture NDN comme cas d'étude car cette architecture nous a semblé la plus aboutie mais également car sa communauté contribue activement à son développement, notamment à travers des programmes dont le code est open-source. La proximité de l'architecture CCN devrait de plus permettre une application directe des résultats obtenus sur NDN. Un autre indicateur démontre clairement la popularité de NDN parmi les architectures ICN : en essayant de comptabiliser le nombre de publications faisant référence aux différentes architectures ICN (par exemple, en notant le nombre de résultats de recherche dans Google Scholar), il apparaît que les deux architectures fédèrent la grande majorité des recherches sur le sujet. Ainsi, quand DONA, NetInf et PURSUIT affichent respectivement publications 565, 706 et 34 publications afférentes, les architectures Content Centric Networking (CCN) et Named Data Networking (NDN), affichent quant à elles respectivement 6380 et 5880 résultats. Ces deux architectures-sœurs sont donc plus intéressantes dans le cadre de nos travaux qui visent un déploiement opérationnel de réseaux orientés contenus.

Dans le chapitre suivant, nous dresserons un état de l'art des différents défis que doivent résoudre les ICN et qui nous semblent critiques pour permettre leur adoption par les fournisseurs d'accès à Internet.

Chapitre 2

Travaux sur les principaux verrous opérationnels des ICN

Sommaire

Nous avons identifié quatre verrous opérationnels principaux qui nous semblent indispensables de résoudre pour que les fournisseurs d'accès à Internet s'intéressent aux architectures ICN. Le premier est lié aux performances de l'architecture qui est un élément essentiel lors de la mise en production d'un réseau ICN. Le second concerne les problèmes de sécurité de l'architecture qui doit être capable de résister aux attaques pour garantir un réseau fonctionnel en toute circonstance aux utilisateurs. Le troisième verrou est lié à l'interopérabilité, car il est primordial de fournir des solutions pour permettre aux applications existantes d'utiliser les ICN sans quoi l'adoption de l'architecture serait ralentie, faute de contenus. Le dernier verrou concerne le déploiement et la gestion de ce type de réseau car la configuration réseau est une tâche complexe qui l'est d'autant plus concernant les ICN car des technologies tierces ou des configurations spécifiques sont utilisées pour permettre leur déploiement sur des infrastructures standards. Ces verrous peuvent être retrouvés en partie (performance, sécurité) dans le RFC [Cha] car celui-ci se focalise sur les questions scientifiques au détriment des questions plus pratiques (interopérabilité, déploiement).

2.1 Problèmes de débits des ICN

L'évaluation des performances n'est pas encore au centre des efforts de recherche actuels du protocole NDN en raison de sa relative jeunesse. Ainsi, à notre connaissance, seuls Yuan *et al.* [YSC12] proposent une étude sur les performances du routeur CCN dans un environnement multi clients/serveurs avec le monitoring et le profilage du débit. Ils exposent le flux opérationnel du routeur et mettent en évidence des problèmes liés à l'évolutivité du programme, comme un flux opérationnel trop complexe, et proposent quelques idées qui permettraient de l'améliorer. Dans [YC13, YC11], ils comparent NDN et HTTP pour un transfert de fichiers lorsque le lien vers

le fournisseur de contenu est limité ou peu fiable. Ainsi, lorsqu'un petit nombre d'utilisateurs demande le même contenu, TCP/IP reste plus efficace. Pour un grand nombre d'utilisateurs, TCP/IP est limité par le lien vers le serveur alors que CCN/NDN garde un temps de transfert semblable grâce à la réplication de contenus dans les caches des nœuds. Les auteurs montrent aussi qu'un nœud CCN/NDN n'est pas encore capable de saturer un lien Gigabit Ethernet de par les faiblesses de l'implémentation. Cependant lorsque la qualité du lien est dégradée (pertes de paquets), le protocole est tout de même capable d'offrir de meilleures performances que TCP/IP.

Certaines études montrent tout de même des tests de performances pour mettre en évidence les avantages de leur propre solution. Guimarães *et al.* [GFT+13] étendent les expériences qu'a fait Van Jacobson [JST+09] dans un environnement virtuel reparti dans tout l'Internet avec leur testbed nommé FITS. Ils montrent les faibles performances de CCN en comparaison avec TCP pour des transferts point-à-point malgré le fait que TCP soit limité par un lien de faible capacité (10 Mbps). HTTP est également souvent mis en concurrence avec CCN/NDN lors de transferts de données car les deux protocoles ont de nombreuses similitudes comme leur schéma de nommage et de communication.

Plusieurs articles se basent sur l'utilisation de filtres de Bloom pour améliorer l'efficacité du routage d'un nœud. Ces structures à base de filtres de Bloom sont utilisées en remplacement de la PIT et/ou de la FIB [WPM+13,LLZM14,QXG+14] pour réduire la quantité de ressources (CPU, RAM) nécessaires au traitement des paquets ICN. Néanmoins, même si ces études montrent des gains significatifs, l'impact sur le réseau des faux positifs ne semble pas pris en compte. Antonio *et al.* ont réalisé une étude du surcoût engendré par l'utilisation de filtres de Bloom dans les ICN. Même si cette étude est basée sur l'utilisation d'un schéma de nommage plat, elle reste potentiellement valide pour NDN même si les tables des routeurs sont de taille moindre. Les conclusions du papier sont que pour des réseaux complexes, l'utilisation de filtres de Bloom peut fortement augmenter l'utilisation de certains liens affectés par les erreurs de routage dues aux faux positifs si la taille du filtre n'est pas suffisamment grande (passage de 192 bits à 384 bits dans leur cas). La mise en place de deux mécanismes palliatifs n'étant pas suffisante : le *fallback*, qui permet de définir une adresse alternative et semble efficace uniquement lorsque le taux de faux positif est élevé, et l'exclusion de préfixe, méthode qui semble efficace peut importe le taux de faux positif mais qui augmente la taille des tables et qui pourrait être contre productive.

D'autres prennent à la fois en compte l'évaluation des performances logicielles et matérielles du transfert de paquets NDN. Won So *et al.* [SNOS13] ont travaillé sur l'implémentation d'un *forwarder* NDN sur des routeurs Cisco avec des modules de services intégrés qui peuvent tirer profit des processeurs multi-cœurs. Ils ont rapporté que leur implémentation peut théoriquement atteindre des débits élevés (20 Gbps) en se basant sur le nombre de paquets *Interest* transmis. Yi Wang *et al.* [WZZ+13] proposent une approche par GPU pour la résolution de noms CCN, une fonction critique d'un routeur CCN/NDN. Leur méthode permet de rechercher et mettre à jour des tables de grande taille dans des délais très courts.

Une bonne partie des travaux sur les performances des réseaux ICN passent par la recherche d'algorithmes efficaces pour permettre une utilisation optimale du réseau et éviter la congestion : Oueslati *et al.* [ORS12] et Carofiglio *et al.* [CGM12], membres du projet CONNECT, ont travaillé sur le contrôle de flux sur deux niveaux : au niveau de l'utilisateur avec l'implémentation de l'algorithme de congestion AIMD et au niveau du routeur avec le partage équitable entre les flux simples et multi-chemins pour éviter la congestion, et une distribution efficace de la bande passante entre les utilisateurs. Junghwan *et al.* proposent un système de gestion de contrôle nommé SMIC [JMT18]. Leur solution est capable d'identifier des flux et d'adapter leur fenêtres de paquets dans un environnement multi-chemins pour mieux tirer parti des capacités du réseau. Cette solution se veut plus adaptée pour du multi-chemins que celle proposée par CONNECT.

Ces solutions se basent sur des fenêtres de congestion mais d'autres solutions se basent sur le taux d'émission de paquets (*rate-based*). Ainsi, Lei *et al.* proposent un premier algorithme de congestion reprenant les bases de RCP (Rate Control Protocol) [Duk08] mais adapté aux spécificités des ICN comme le routage de proche en proche et la symétrie des requêtes, les auteurs montrent que leur protocole permet de meilleurs résultats qu'avec l'algorithme AIMD. Milad *et al.* proposent un algorithme de congestion nommé MIRCC [MAGO16], qui se veut une meilleure alternative au précédent avec une convergence plus rapide tout en étant utilisable dans des scénarios multi-chemins. Comme on peut le remarquer, les travaux de recherche sur la résolution de la congestion dans les réseau ICN sont nombreux et semblent aussi bien traiter le problème au niveau des nœuds du réseau que de l'utilisateur.

Cenk *et al.* [GKS+18] proposent une comparaison entre NDN, ainsi que leur propre proposition de modèle publish-subcribe sur NDN [GKSW18a, GKSW18b], et deux protocoles IP : CoAP et MQTT. Leurs expériences montrent que pour des réseaux simples et fiables, les différents protocoles montrent des comportements similaires mais les protocoles IP restent les plus efficaces. Mais dans le cas de réseaux avec plusieurs sauts ou avec des erreurs de transmission, les protocoles ICN permettent l'acheminement des données avec peu de retransmission. Toujours dans le domaine de l'IoT, ils proposent une façon de compresser les paquets *Interest* et *Data* nommée ICN-LoWPAN [GKSW18c], grâce à l'utilisation de compression *stateless* et *stateful*. Ils montrent une réduction du nombre d'octets envoyés de 36% pour les paquets *Interest* et de 76% pour les paquets *Data* sur un panel de 1000 requêtes.

La recherche s'est aussi intéressée très tôt à la problématique du cache dans les architecture ICN [ZLL13, ZLZ15], qui est connexe à celle du débit dans le réseau. Une bonne utilisation du cache permet de limiter la congestion puisque les liens sont moins sollicités et d'accélérer le transfert des données car le cache étant plus proche de l'utilisateur, il est supposé plus rapide que le fournisseur de contenus. Les articles traitant du sujet couvrent un très large éventail de propositions, cherchant à optimiser le placement des nœuds capables de mettre en cache les contenus, mais aussi les stratégies de mise en cache et de remplacement des contenus déjà présents dans ceux-ci, voire des notions plus complexes comme la collaboration entre différentes instances pour réduire la redondance des contenus en cache.

Ainsi, il apparaît que la recherche s'intéresse majoritairement à l'efficacité du routage, du cache et à la congestion des ICN mais n'accorde que peu d'attention aux performances applicatives offertes par le réseau. Par exemple, à notre connaissance, il ne semble pas y avoir eu d'études de performances sur les débits pouvant être offerts par un fournisseur de contenus, ce qui pourrait aussi être un élément limitant du réseau notamment à cause du coût important de la signature des paquets. C'est précisément ce que nous nous proposons d'étudier dans le chapitre 3.

2.2 Problèmes de sécurité des ICN

L'article [AHZ15] classes les différentes attaques possibles sur les ICN et les classe en quatre catégories : nommage, routage, cache et autres. Certaines de ces attaques sont communes avec les réseaux existants mais une grande partie reste spécifique aux ICN et semblent être majoritairement des attaques de type déni de service [GTUZ13, AZ15]. Dans la première catégorie, les auteurs listent deux types d'attaques : *watchlist* et *sniffing*. Pour être réalisées, il faut pouvoir compromettre un nœud dans le réseau. Les attaques basées sur les noms consistent à filtrer ou surveiller les contenus demandés par un ou plusieurs utilisateurs, selon une liste établie de noms ou

de mots clés au niveau d'un nœud corrompu en supprimant ou non les requêtes et/ou les réponses pour ces contenus. Pour les attaques liées au routage, celles-ci sont majoritairement basées sur des variantes de l'attaque par inondation de paquets *Interest* ou IFA, une attaque par déni de service considérée comme la plus critique pour une architecture ICN. Par cette attaque, un attaquant peut facilement dégrader les performances d'un nœud du réseau ou d'une source en le/la surchargeant (*resource exhaustion*). Il lui suffit d'envoyer un grand nombre de paquets *Interest* sur des contenus existant ou non (focalisé sur un préfixe pour une source). Cela va surcharger les PIT des nœuds, qui vont prendre à minima plus de temps pour la résolution des noms et risquent de jeter des requêtes quand la taille maximale de la PIT est atteinte. C'est l'inconvénient d'avoir une structure de données *stateful* au cœur du routeur NDN/CCN. Cette attaque peut être amplifiée par les réémissions de paquets *Interest* des utilisateurs légitimes lorsque leurs demandes expireront. L'IFA est une attaque jugée critique et est en cours de résolution grâce à une recherche active sur sa caractérisation [TZLZ13, WCZ+14, NCDR15, DLC+16, MND+16] et la proposition de plusieurs contre-mesures pour atténuer ses effets [AMM+13, WZQ+13, CCGT13, SWS15, XLW+16].

Il est possible de réaliser une attaque similaire sans inondation. Pour cela un attaquant va envoyer des requêtes qui ciblent des contenus avec un long temps de réponse (*piling requests*) [WSV13]. Il est aussi possible de polluer la FIB d'un nœud en envoyant des demandes de souscription de route invalides. Ainsi, cette attaque permet comme précédemment de consommer inutilement des ressources et peut rediriger des contenus vers ces routes qui ne répondront pas aux requêtes des utilisateurs. Dans le cas où un routeur vérifierait les signatures des paquets avant des les mettre dans son cache, un attaquant peut faire coopérer deux entités, dont l'une sera chargée de récupérer les contenus publiés par la seconde. Ici le mauvais fournisseur de contenus va signer ses contenus avec un important nombre de clés différentes ce qui va ralentir le routeur car il devra passer beaucoup de temps à récupérer les clés et vérifier les signatures de ces paquets. Ces attaques sont évitables en ne vérifiant pas les signatures en chemin pour les contenus mais seulement pour les annonces de route, ce qui est le comportement par défaut pressenti pour NDN.

Les autres types d'attaques consistent essentiellement à manipuler le trafic en modifiant des paquets, en faisant du rejeu ou encore en répondant avec de faux paquets. Il est aussi possible d'accéder à des contenus spécifiques à certains utilisateurs grâce à la manipulation des caches, ou encore deviner la clé privée utilisée par un fournisseur de contenus grâce à l'abondance de paquets signés avec cette clé dans les caches si le protocole de chiffrement n'est pas assez robuste.

Le cache de contenus est une fonction importante des ICN. Cependant, cela expose aussi le réseau à d'autres menaces ciblant cette table en particulier [CDCKU13, Lau10, AKRS11, ACG+13]. Des attaquants peuvent exploiter les caches pour obtenir des informations non autorisées ou pour saboter le système. Outre la CPA (*Content Poisoning Attack*) ou attaque par empoisonnement de contenus, les attaques liées à la mise en cache de contenus peuvent être classées en deux catégories : l'analyse temporelle et la pollution du cache. Dans la plupart des papiers relatifs à ce sujet [ACG+13, NH12, MZS+13], l'analyse temporelle est décrite comme une attaque exploitant la différence entre le délai de réception de paquets *Data* venant du fournisseur de contenus et les copies venant du cache pour connaître les récentes requêtes des utilisateurs, violant ainsi leur vie privée. Pour la pollution de cache, un attaquant peut forcer les caches à stocker les contenus de façon irrégulière afin de réduire les performances de ces caches pour des utilisateurs légitimes [XWW12, CGT13, RRAG14]. Cette attaque a largement été étudiée pour les réseaux IP, principalement pour le cache de contenus web. L'objectif principal de cette attaque est de dégrader les performances du cache, en envoyant une grande quantité de demandes de contenus impopulaires. Les contenus populaires seront moins fréquemment trouvés dans les caches, donc plus de demandes seront acheminées en amont, ce qui augmentera le trafic dans le réseau. Les

auteurs de [DGCK08] proposent deux classes génériques d'attaques par pollution de caches : *false-locality* et *locality-disruption*. Ces attaques consistent soit à cibler un petit nombre de contenus non populaires pour la première, soit un grand nombre pour la seconde. Dans le premier cas, le but est d'augmenter artificiellement la popularité de ces contenus pour que les caches remplacent des contenus légitimement populaires par ceux-ci. Dans le second cas, le but est de réduire les écarts dans la distribution de la popularité des contenus pour réduire la présence des contenus populaires dans les caches.

La CPA, au contraire de la pollution de cache, ne se contente pas uniquement de requêtes mais vise à mettre réellement en cache des contenus inappropriés. Ainsi, les paquets *Interest* légitimes ont toujours une réponse, mais par des paquets *Data* inappropriés qui peuvent être injectés dans le réseau par des routeurs compromis ou indirectement par une collaboration entre mauvais fournisseurs de contenus et utilisateurs. Ces paquets *Data* ont toujours un nom valide mais leur contenus a été altérés. Ce genre d'attaque exploite les caches du réseau NDN pour diffuser de mauvais contenus au plus grand nombre d'utilisateurs possible. L'attaquant est susceptible de forger des paquets empoisonnés avec des noms de contenus populaires pour augmenter l'effet de l'attaque. Dans [GTUZ13], les auteurs indiquent deux types de paquet *Data* empoisonnés : (1) *Corrupted Data* et (2) *Fake Data*. Le contenu est dans les deux cas modifié. Dans le premier cas, le mauvais fournisseur de contenus ne possède pas les bonnes informations de signature pour correctement générer la signature du contenu modifié. Ce type de paquet est plus enclin à être créé mais peut être détecté en vérifiant la signature du paquet qui échouera au niveau de l'utilisateur. Dans le deuxième cas, le mauvais fournisseur de contenus possède les informations de signature pour signer correctement les paquets. Néanmoins, ils sont difficiles à créer car il faut pour cela réussir à récupérer la clé privée d'un fournisseur de contenus légitime, ce qui est une tâche difficile, mais les paquets forgés seront alors impossible à détecter.

Les solution proposées à ce jour pour détecter et pallier la CPA sont restreintes en nombre et peuvent être divisées en deux catégories : la vérification par le réseau et la rétroaction (ou exclusion). Les routeurs peuvent pallier la CPA en vérifiant les paquets *Data* avant de les transmettre. Mais en réalité, cela est impossible à mettre en place au niveau des routeurs à cause du coût calculatoire élevé de cette opération [GTU14a] créant ainsi un autre moyen d'attaquer les routeurs, comme mentionné précédemment. Par conséquent, le principale objectif de la première approche consiste à réduire le coût de cette vérification au niveau des routeurs, en changeant leur routine de vérification [BDCBM13, KNBY15, RRAG14, GTU14b], ou politique de cache [KGZ15] tout en préservant sa résistance contre la CPA. Une solution typique de cette catégorie est la proposition de Kim et al. [KNBY15]. Dans cette approche, un routeur accepte de mettre en cache tous les paquets *Data* qu'il transmet, mais ne les vérifie que lorsque qu'ils sont réutilisés. Les paquets *Data* qui passent la vérification de signature sont transmis sans vérification supplémentaire. Cela réduit fortement la charge du routeur tout en maintenant la vérification de signature pour les contenus populaires. Mais cette solution ne résout le problème que localement car les mauvais clients peuvent toujours réémettre de nouveaux faux paquets à insérer dans le cache, ou augmenter la charge des routeurs en envoyant des paquets *Interest* pour générer des hits sur des contenus peu populaires. D'un autre côté, les solutions de la seconde catégorie [DP16, GTU14a] se basent sur le fait qu'un utilisateur peut exploiter le champ *Exclude* pour éviter les paquets *Data* qu'il ne souhaite pas. Le classement des contenus [GTU14a] est une solution typique de cette catégorie dans laquelle un routeur ordonne les copies en cache des paquets *Data* sur trois caractéristiques traduisant le degré d'exclusion des utilisateurs : nombre d'exclusions, temps de distribution et le nombre d'exclusions par *Faces* entrantes. Les copies en cache avec un haut rang sont plus à même d'être transmises aux utilisateurs. Néanmoins, une des faiblesses est que cela se base sur les exclusions émises par les clients, et de ce fait ne sont pas fiables, le mécanisme

pouvant lui-même être détourné à des fins malveillantes. De plus, l'exclusion est aussi utilisée pour l'exploration de contenus NDN. Ainsi, des paquets *Data* valides pourraient être considérés comme non-valides quand les utilisateurs l'excluent pour récupérer un contenu différent.

En plus de ces limites, les travaux précédents sur la CPA partagent quelques inconvénients. Le premier est le manque de rigueur dans l'étude de l'impact de la CPA. Comme les auteurs évaluent habituellement la CPA avec leurs propres solutions, la compréhension de ce phénomène est partielle et biaisée pour mettre l'accent sur les solutions proposées. De plus, les résultats ne sont ni réutilisables, ni comparables. En second, les scénarios de simulations se basent sur une attaque ponctuelle dans laquelle les clients s'arrêtent le plus souvent après avoir reçu un paquet *Data* légitime, alors que la CPA est plus susceptible d'être utilisée comme un flux continu. Ainsi, cela laisse des zones d'ombres sur ce phénomène. Pour finir, la plupart des travaux précédents surestiment la CPA avec des comportements irréalistes. Par exemple, ils n'utilisent pas le champ *Exclude* des paquets *Interest* pour éviter de recevoir de mauvais paquets *Data*, ou considèrent des caches pré-pollués avec des taux de mauvais paquets *Data* très élevés et peu réalistes. Bien qu'il y ait des explications sur les scénarios d'attaques et sur la façon dont les mauvais paquets *Data* sont insérés dans les caches, elles sont insuffisantes pour expliquer pourquoi la CPA peut atteindre un tel pourcentage.

Ces inconvénients soulèvent la nécessité d'une étude neutre et approfondie pour caractériser rigoureusement la CPA avant de développer tout mécanisme de détection ou de protection de cette attaque. Pour ce faire, nous proposons dans le chapitre 4 une évaluation exhaustive de l'impact de la CPA sur tous les composants d'un réseau déployant de véritables nœuds NDN avec l'utilisation d'un protocole expérimental détaillé, ainsi que la compréhension de sa mise en œuvre et des analyses de cette attaque.

2.3 Problème d'interopérabilité des ICN avec les applications existantes

Comme le déploiement des réseaux ICN va principalement être dirigé par les fournisseurs d'accès à Internet, il est peu probable que ce type de réseaux soit déployés à grande échelle dès le départ. D'un autre côté, les utilisateurs et les développeurs d'applications sont souvent réfractaires aux changements et il y a peu de chance que les réseaux ICN soient utilisés par une grande partie de la population à moyen terme comme c'est déjà le cas aujourd'hui avec IPv6, qui arrive difficilement à 25% d'utilisation alors que le protocole à maintenant 20 ans et que son déploiement a déjà quelques années. Il est donc important de veiller à ce que les applications actuelles puissent être utilisables sur les ICN afin de faciliter leur adoption. Différentes approches ont été explorées pour transporter du trafic réel sur des réseaux NDN ou CCN. Le plus souvent, celles-ci cherchent à prendre en charge le trafic web qui représente la majorité des données qui transitent actuellement sur Internet. Le streaming vidéo étant en tête des usages avec près de 75% du trafic global en 2017 dont les trois quarts sont sur le web (Youtube, etc...) et ce type de trafic devrait représenter 82% du trafic global d'ici 2022 selon les prévisions annuelles de Cisco [Cis18], les autres contenus web représentant 17% du trafic global. Une interopérabilité HTTP/ICN permettrait donc d'amener progressivement jusqu'à trois quart des données transitant sur Internet vers les ICN. Ainsi, nous avons identifié deux façons de transporter du trafic web sur ICN.

Une première approche repose sur l'utilisation de *gateways*. Des ingénieurs de CableLabs [WR16] ont ainsi identifié deux technologies essentielles constituant leur architecture permettant

de migrer les CDN sur réseau ICN, à savoir la traduction HTTP/CCN et l'utilisation de tunnels CCN/IP. Ils ne proposent cependant qu'une description fonctionnelle de haut niveau des services attendus par leur *gateway*. Sen Wang et al [WBW+12] ont implanté de telles *gateways* qui vont transmettre le trafic d'un réseau IP vers un réseau CCN en adaptant les requêtes HTTP des utilisateurs pour être compatibles avec les paquets *Interest* du réseau CCN. Ils visent à introduire du trafic web réel dans les réseaux CCN mais en considérant la façon dont leur solution génère les noms des paquets *Interest*, celle-ci souffre de plusieurs limites. La première est qu'elle n'est pas compatible avec des scénarios où les *gateways* auraient à communiquer avec un fournisseurs de contenus natif. En soit, ils donnent un exemple de requête HTTP GET où le nom de la *gateway* de sortie est explicitement donné dans le nom du paquet ou précédé par « /default » pour signifier « la plus proche ». De plus, la façon de traiter les requêtes HTTP POST n'est pas très claire puisque aucune information supplémentaire n'est donnée lors de la génération du nom du paquet, mais le nom du paquet utilisé pour les requêtes HTTP GET est insuffisant pour transporter une requête HTTP POST car la *gateway* de sortie ne peut pas identifier par quelle *gateway* d'entrée la requête a été générée. Sans cette information, il n'est pas possible de récupérer le contenu de la requête HTTP POST. Pour finir, les auteurs ajoutent au paquet *Interest* un nouveau champ, nommé « metadata », dans le but de transporter l'entête de la requête HTTP. Cette modification peut poser quelques problèmes au niveau de la PIT des nœuds CCN car le parcours de la table ne considère que le nom du paquet pour réaliser une correspondance. Ainsi, comme il est possible de récupérer des contenus différents via une même URL en HTTP, il serait possible d'avoir des contenus non désirés, car en cas d'agrégation de requêtes dans la PIT, un même contenu pourrait répondre à des requêtes différentes sur un même nom CCN. Une autre approche plus générique, qui se base aussi sur des *gateways* a été proposée par Ilya Moiseenko and Dave Oran [MO16]. Ils ont réussi à transporter le protocole TCP sur NDN, et en réalisant cela, ils permettent de transporter n'importe lequel des protocoles utilisant TCP comme couche de transport, dont HTTP(S). Bien que leurs travaux soient pratiques et qu'ils puissent conduire à faire grandement avancer l'adoption et le déploiement de réseau NDN, leur solution reste générique et ne permet pas d'utiliser certaines fonctionnalités des réseaux ICN, et plus spécifiquement le cache qui est l'une des fonctions majeure de ce type de protocole car il est presque impossible de profiter efficacement du cache en ne considérant qu'une connexion TCP. En effet, il serait au plus limité à un seul utilisateur car les données ne seraient pas réutilisables par d'autres car le cache manipule des données de trop bas niveau et liées aux spécificités du protocole TCP.

Une seconde approche est de donner aux applications la capacité de communiquer directement sur les réseaux ICN. Pour le cas du web, il existe une librairie NDN en JavaScript, nommée NDN.JS [STC+13], et qui est aussi disponible comme plugin Firefox et permet aux utilisateurs d'utiliser un réseau NDN en tapant des URL de la forme « ndn :// » ou encore DASC [LGP+13] une adaptation de DASH sur CCN qui reste capable d'utiliser le réseau traditionnel lorsqu'un contenu n'est pas disponible via CCN. Une autre approche a été prise par Xiuquan Qiao et al [QNP+15], ils ont conçu un navigateur web complet, nommé « NDNBrowser », qui peut gérer les requêtes HTTP aussi bien sur CCN que IP [QNT+14]. En gérant nativement le protocole CCN, les auteurs montrent un gain substantiel de performances comparé aux solutions présentées précédemment. Ils ont aussi modifié un serveur Tomcat pour lui permettre de communiquer avec le protocole CCN en plus d'IP [QNT+14]. Les auteurs pensent qu'en fournissant à la fois un serveur et un navigateur web qui communiquent aussi bien sur IP que CCN, la transition du trafic web vers les ICN sera simplifiée.

D'autres protocoles ont pu être adaptés sur les ICN comme VoCCN [JSB+09] ou plus spécifiquement pour la diffusion de contenus vidéo, des adaptations de protocoles existant comme

HLS [XCCC12] ou la mise en place de protocoles spécifiques aux ICN comme NDNVideo [Bur13] pour n'en citer que quelques uns. Par contre, les solutions reprenant des protocoles existant ne tiennent pas forcement compte de l'interopérabilité avec les réseaux IP. Ainsi, il existe deux grandes façons de rendre interopérables des réseaux différents pour un même protocole : au niveau du client, en développant des applications à part entière ou des extensions sous forme de plugins pour les applications qui le supportent, ou au niveau du réseau avec le déploiement de *gateways* permettant d'adapter le protocole aux spécificités de l'architecture.

Ainsi, nous pensons que l'usage de *gateways* HTTP/NDN reste la meilleure solution actuellement pour transporter du trafic web sur NDN car cela permet aux FAI de créer des petit îlots pour réaliser des expériences réelles de leur côté et, si ils sont satisfaits des résultats, ils pourront incrémentalement augmenter la taille de ces îlots jusqu'à couvrir tout leur réseau. Parmi les solution à base de *gateways*, celle permettant de transporter TCP, malgré sa grande flexibilité, ne permet pas de profiter pleinement des fonctionnalités des ICN. Ainsi il semble plus judicieux de développer des *gateways* spécialisées pour un seul protocole pour permettre une conversion optimale. Dans le cas du web, la seule solution existante nous semble lacunaire et c'est pourquoi nous proposons notre propre solution de *gateways* dans le chapitre 5, qui cherche à exploiter au mieux les fonctionnalités propres de NDN, mais aussi permettre la communication entre des entités IP et NDN.

2.4 Difficultés de déploiement des ICN

La recherche a déjà exploré différentes façons pour déployer les ICN en parallèle de réseaux IP. La plupart du temps, le trafic ICN est transporté au dessus d'IP avec des composants réseau qui sont plus ou moins conscients du protocole ICN. Ainsi, la majorité des propositions s'appuient fortement sur l'utilisation de SDN (Software-Defined Networking) pour permettre aux réseaux IP de pouvoir router le trafic ICN sans encombre. Différentes solutions ont été proposées selon le degré de compréhension du réseau : Vahlenkamp et al. [VSKS13] proposent une solution où les routeurs du réseau ne sont pas conscients du trafic ICN mais sont au minimum capables de détecter ses paquets. Ainsi, les paquets non résolus sont envoyés vers un contrôleur SDN qui est capable de gérer les protocoles ICN et poussera des règles vers les routeurs du réseau pour qu'ils puissent router les paquets ICN de la bonne manière. D'autres solutions ont été proposées lorsque seulement une partie du réseau est consciente du trafic ICN. Par exemple dans le cas de CONET [SBMD+13], les routeurs de bordure sont capables de comprendre le trafic ICN en se basant sur des règles d'étiquetage envoyées par le contrôleur SDN aux routeurs IP. Cette solution étend OpenFlow en plus d'utiliser les options et la charge utile d'IP pour le transport d'informations relatives aux ICN. D'une manière similaire, lorsque des routeurs ICN ne sont pas restreints à la bordure du réseau, NDNFlow [vAK15] étend OpenFlow avec un canal de contrôle dédié pour la communication et la résolution des paquets ICN pour une approche plus propre. Des règles sont envoyées vers les routeurs capables de gérer le trafic ICN pour créer des tunnels IP entre eux. [NST13] propose un protocole au niveau d'un routeur pour transmettre des paquets ICN sur un réseau IP. Pour cela le protocole va calculer un hash du nom du paquet est l'utiliser comme valeur pour les champs IP/port des paquets IP.

Hybrid ICN [Net] va encore plus loin dans l'usage (le détournement) des champs du protocole IP mais ne repose pas sur SDN pour le routage. Ici, il n'est pas question d'adapter un protocole ICN mais de réutiliser ceux que nous utilisons aujourd'hui (IPv6, TCP, IPsec) de façon à intégrer la logique des architectures ICN. Ainsi, l'entête IPv6 voit son champ IP de destination utilisé

d'une manière différente pour une requête : les 64 premiers bits sont un préfixe IPv6 routable globalement et les 64 derniers bits représentent un identifiant de la donnée, une réponse inversera IP source et IP de destination. Une application pourra donc utiliser plusieurs adresses en même temps puisque l'on aura une IP par contenu. Pour l'entête TCP, les champs « Numéro de séquence » et « Numéro d'acquittement » vont être utilisés pour respectivement le numéro de segment de la donnée et un label du chemin entre l'utilisateur et l'entité ayant répondu à sa requête. Le champ réservé ainsi que les trois premiers *flags* deviennent un unique champ de 6 bits servant à amplifier la valeur du champ de durée de vie. Les flags URG et PSH sont respectivement utilisés pour signaler que le paquet contient un *manifest* et/ou une signature qui sera contenu dans un entête IPsec. Le champ de taille de fenêtre sera utilisé pour la détection et la récupération de paquets et pour finir le champ « pointeur de données urgente » sera utilisé pour définir la durée de vie du paquet. Cette façon de composer les paquets leur permet d'être routés par des routeurs IPv6 classiques. Il est possible de déployer des routeurs « améliorés » en ajoutant un cache local comme une fonction de pré-traitement qui va contenir aussi bien les requêtes que les réponses pendant de faibles durées. Lors de la réception d'une requête, le routeur va regarder si il y a une réponse ou d'autres requêtes sur le même contenu, la correspondance étant stricte (préfixe + suffixe). Dans le cas d'une correspondance pour une réponse, le routeur va changer l'adresse de destination par l'adresse source de la requête avant de la transmettre. Dans le cas d'une correspondance pour une requête, si l'adresse source est identique, le paquet est considéré comme dupliqué, sinon il sera stocké dans le cache et ne sera pas routé. Pour finir, si il n'y a aucune correspondance, le routeur va stocker la requête dans le cache et va remplacer l'adresse source par la sienne avant de transmettre le paquet. Lors de la réception d'une réponse, si il y a une correspondance, le paquet va être cloné autant de fois qu'il y a de requêtes dans le cache et pour chacun d'eux l'adresse de destination sera remplacée par l'adresse source de la requête, sinon le paquet est écarté. Cette façon de faire permet un déploiement quasi-immédiat d'une architecture orientée contenus car il ne modifie pas l'infrastructure réseau existante. De plus, il existe une implémentation d'une couche de transport mettant en avant hybrid ICN [SMC18]. Elle est développée avec VPP et DPDK pour obtenir de bonnes performances et permet une abstraction entre les couches réseau et applicative ce qui libère les développeurs de concepts liés à l'utilisation des ICN comme la signature, la segmentation etc...

Enfin, quelques initiatives récentes pour les réseaux ICN tirent parti de l'architecture NFV (*Network Function Virtualization*) et des technologies associées [LC15, MSG+16] pour prendre en charge le déploiement des ICN. NFV est un paradigme, dont le premier rapport a été publié en 2012 [CCW+12] et qui se veut une alternative à l'utilisation actuelle de matériel réseau, car il existe de nombreux équipements réseau et chacun a un usage spécifique. Ces équipements sont onéreux et difficilement extensibles, ce qui limite fortement l'innovation dans le cœur des réseaux. Le principe de NFV est de pouvoir chaîner des composants, appelés *Virtual Network Function* (VFN), qui regroupent une ou plusieurs fonctions réseau de base pour pouvoir créer un ensemble formant un service. Il existe déjà des applications capables de reproduire des fonctions réseau comme les bridges ou l'ip forwarding couplé avec la table de routage de la machine. Cela reste cependant très basique et il faudra encore quelques années pour que des entreprises développent des logiciels aussi personnalisables que ce que l'on peut trouver dans les matériels des équipementiers, comme ceux de CISCO. De plus NFV ne s'arrête pas à l'application. Il apporte tout un principe basé sur la virtualisation de ces composants ainsi que leur orchestration. Le fait de pouvoir chaîner facilement des fonctions réseau permettra aux entreprises de proposer des services plus rapidement sur le marché et de répondre plus vite aux variations de charge grâce à une mise à l'échelle horizontale des VNF, reprenant ainsi un des principes clés qui a fait le succès du *cloud computing*. vICN [SMA+17] a la particularité de proposer un Framework

de programmation orienté objet générique utilisant une représentation précise des différentes ressources utilisées pour construire des topologies de réseaux ICN virtualisés. Par rapport à la philosophie du projet DOCTOR, les deux approches ont des similitudes comme le fait d'utiliser des conteneurs et des commutateurs virtuels pour l'infrastructure de réseau virtuel (VNI) et le fait de suivre les directives de l'ETSI concernant la gestion et l'orchestration de NFV [Gro14]. MiniCCNx [CRMa13] utilise également des technologies similaires mais offre un environnement d'émulation pour l'expérimentation plutôt que pour construire un réseau de production, tandis que [GFT+13] propose son *Futur Internet Testbed* incluant des aspects de sécurité pour effectuer des expériences sur ICN.

Proche du concept NFV mais dans une perspective 5G, le découpage de réseau (*network slicing*), où plusieurs protocoles cohabitent dans une même infrastructure grâce à la virtualisation, est envisagé par l'IRTF ICNRG [RTKR18] comme un scénario possible pour un déploiement futur d'ICN. Les auteurs de 5G-ICN [RCA+17] proposent une telle architecture qui permet la mobilité en tant que service, mais ils se concentrent principalement sur le plan de données et nous manquons de détails sur le plan de contrôle. Pour améliorer les performances des réseaux NFV, les auteurs de [PGL+16] appellent au support de l'accélération matérielle et proposent un plan de données programmable (PDP) qui permet de personnaliser à la volée les piles L2 à L7 (modèle OSI) pour intégrer des accélérateurs matériels. Leur exemple instancie un routeur ICN accéléré qui est décomposé en six modules mais ils ne sont pas conçus pour être autonomes et ne sont pas managés.

Une grande partie des solutions proposées se basent sur l'usage d'un contrôleur SDN pour permettre au réseau de router correctement les ICN. Selon le degré de compréhension des nœuds du réseau, les solutions sont plus ou moins complexes et peuvent aller jusqu'à altérer les entêtes des paquets. De plus, les fonctionnalités spécifiques aux ICN sont plus ou moins réalisables selon les propositions. D'un autre côté, hybrid ICN permet un routage sans composant additionnel en dérivant certains champs des entêtes de protocoles IP pour répondre aux spécificités d'un protocole ICN. D'autres solutions se basent sur la virtualisation pour faire cohabiter les deux réseaux sur une même infrastructure. Le paradigme NFV va encore plus loin en proposant un framework pour le déploiement de fonctions réseau virtualisées, peu de solutions utilisent ce paradigme car il est encore récent et semble grandement simplifier le déploiement de réseaux existants ou futurs.

Nous pensons que la virtualisation réseau grâce à NFV permet un déploiement propre des ICN et de capitaliser sur du matériel générique tout en conservant les propriétés du protocole. Cependant, il reste à créer les VNF permettant de déployer une architecture ICN et surtout de les orchestrer. Ainsi, nous proposons une solution à base de microservices et utilisant les concepts de NVF pour répondre à cette problématique du déploiement d'un ICN dans le chapitre 6.

2.5 Conclusion

A l'issue de l'état de l'art qui fut guidé par les quatre verrous opérationnels suivants : les performances, la sécurité, l'interopérabilité et le déploiement des réseaux ICN, nous faisons les constats et les propositions suivantes.

Concernant les performances des réseaux, certains aspects tels que le contrôle de congestion, les politiques de mise en cache ou encore les performances du routeur semblent bien traités et aboutis. En revanche, peu de contributions traitent le problème du débit de bout en bout dans un réseau ICN. Dans cette thèse, nous nous proposons donc d'étudier le débit que peut générer un fournisseur de contenus et l'impact des différentes configurations du protocole sur la

machine l'exécutant. Nous allons jusqu'à proposer des améliorations permettant une génération de paquets plus efficace pour un fournisseur de contenus. Ceci est traité dans le chapitre 3.

Les attaques ciblant les architectures ICN semblent majoritairement des attaques par déni de service. Parmi elles, l'IFA est considérée comme la plus dangereuse, de par sa simplicité et son efficacité. Beaucoup d'efforts de recherche ont été faits pour caractériser et atténuer ses effets et l'on peut désormais considérer cette attaque comme en court de résolution, voire résolue. La CPA est considérée comme la deuxième attaque la plus importante pour les ICN mais elle n'a pas reçu autant d'attention que la première, des solutions sont proposées mais les études sont souvent limitées et spécifiques à la solution proposée. Nous proposons donc dans cette thèse une étude approfondie de la CPA ainsi que sa caractérisation. Cette étude nous a permis de découvrir une faille dans le traitement des paquets que nous avons corrigée. Ces travaux sont présentés dans le chapitre 4.

Pour permettre l'interopérabilité entre les réseaux existants et les architectures ICN, deux solutions sont principalement proposées : l'adaptation de l'application pour un support natif, ou la mise en place de *gateways* se chargeant d'adapter un protocole particulier. La première solution est la plus efficace mais nécessite qu'un réseau ICN soit déjà déployé pour pouvoir fonctionner, alors que la deuxième ne nécessite qu'un réseau partiel sous forme d'îlots. Parmi les propositions de *gateways*, la plus aboutie est probablement la solution de transporter TCP dans NDN mais elle pose aussi le problème de l'utilisation limitée des fonctionnalités propres aux architecture ICN. Dans cette thèse, nous proposons notre propre solution de *gateways* pour HTTP qui cherche à combler les lacunes des solutions précédentes, notamment en respectant le protocole et en cherchant à utiliser au mieux les fonctionnalités des architectures ICN, en particulier une mise en cache mutualisée des contenus web. Cette contribution est traitée dans le chapitre 5.

Enfin, concernant le déploiement d'une architecture ICN, deux grandes tendances ont été explorées : utiliser les réseaux existants pour transporter les paquets ICN ou déployer une architecture ICN à côté des réseaux existants. Dans le premier cas, les propositions se basent beaucoup sur l'ajout d'un contrôleur SDN pour palier les lacunes du réseau mais elles semblent complexes dans leur mise en œuvre et limitées dans leur compréhension du protocole ICN, et donc dans les fonctionnalités supportées. Dans le deuxième cas, le déploiement est réalisé grâce à la virtualisation et parfois en utilisant le paradigme NVF. NFV étant récent, peu de solutions l'utilisent. Il manque notamment la définition de VNF propres aux ICN et de l'orchestrateur permettant leur bonne gestion. C'est pourquoi nous proposons dans le chapitre 6 une architecture à base de microservices ainsi que leur manager qui reposent sur les concepts de NFV.

Deuxième partie

Performances et sécurité

Chapitre 3

Évaluation de performances d'un fournisseur de contenus NDN

Sommaire

3.1 Introduction

Comme nous l'avons vu dans le chapitre précédent, la majorité du trafic Internet vient de services diffusant des vidéos comme Netflix ou YouTube, mais les protocoles actuels ne sont pas prévus pour la diffusion de masse à l'échelle d'Internet. Par exemple, si l'on reprend l'exemple des vidéos en streaming, si une vidéo est regardée par 1000 spectateurs, le serveur distribuant cette vidéo devra l'envoyer simultanément aux 1000 spectateurs et générera donc un important trafic [9].

Le protocole NDN a été pensé avec l'intention de réduire la congestion des réseaux en regroupant les paquets *Interest* pour un même contenu directement au niveau du réseau et ne transmettant que la première demande, dans un laps de temps donné, au serveur (plus de possibles retransmissions). Ainsi le serveur n'aura besoin que de répondre à cet unique paquet et le réseau se chargera de le transmettre à tous ceux qui en ont fait la demande. NDN propose aussi d'autres fonctions avancées comme un système de cache sur chaque nœud du réseau pour réutiliser les contenus récemment transmis et/ou populaires. Les paquets NDN contenant de

9. nous sommes ici dans le cas d'un fournisseur de contenu OTT, ne pouvant utiliser de multicast au sein d'un AS.

l'information, nommés (*Data*), doivent être signés pour garantir l'authenticité de l'émetteur du paquet ou au moins contenir un hash (SHA-256) pour en garantir l'intégrité. Cependant en signant tous les paquets, les échanges avec le protocole NDN peuvent devenir très demandeurs en ressources CPU car chaque paquet *Data* généré devra être signé. Cela peut devenir critique dans le cas d'applications qui fournissent des contenus en temps réel (vidéo en direct, jeu en ligne, VoIP, etc.) qui, en étant générés à la volée, ne peuvent donc pas être signés à l'avance.

La plupart des papiers de recherche se concentrent sur les performances du cache NDN mais aucun ne considère les performances de la génération de paquets *Data* alors qu'il est aussi important de les générer efficacement que de les redistribuer dans le cas des applications temps réel comme les sites de live streaming (Twitch [10], Periscope [11], etc.). Dans ce chapitre, nous donnons une première évaluation du débit que peut atteindre un fournisseur de contenus NDN tout en mesurant sa consommation CPU dans plusieurs scénarios, grâce à un outil open source que nous avons développé : NDNperf. À notre connaissance, aucune étude à ce jour ne s'est encore penchée sur la question de l'impact de la génération de signatures pour le protocole NDN sur les performances d'un fournisseur de contenus, ce que nous nous proposons de faire dans les prochaines sections.

Durant ce chapitre nous parlerons dans un premier temps de l'outil que nous avons développé, puis de l'étude de performances que nous avons réalisée avec celui-ci. Pour finir nous proposerons de possibles améliorations pour réduire le coût de la signature d'un paquet et l'évaluation de leur efficacité, puis nous conclurons ce chapitre.

3.2 Environnement expérimental

Nous avons développé NDNperf, un outil distribué en open source, pour l'évaluation de performances et le dimensionnement de serveur NDN. Cet outil permet de mesurer le débit qu'un serveur est capable de fournir quand il a besoin de générer de nouveaux paquets *Data*. Cet outil est similaire à iPerf qui lui aussi, a besoin d'un duo client/serveur pour réaliser ses mesures de performances. NDNperf essaie de minimiser le nombre d'instructions entre la réception et l'envoi d'un paquet pour obtenir la mesure la plus fiable possible. Le logiciel a été développé dans deux langages, Java et C++, mais la version C++ est préférée car plus rapide alors que la version Java existe plus pour le développement de fonctionnalités expérimentales. Les fonctions mises en avant de NDNperf sont :

- le rapport périodique des performances par le client et le serveur (débit, latence, temps nécessaire pour traiter un paquet, etc.) ;
- un serveur multithread (un thread pour la boucle d'écoute et N threads pour la génération de paquets) ;
- la capacité d'utiliser tous les types de signature disponibles dans la librairie NDN, ainsi que le choix de la taille de la clé ;
- la possibilité de choisir la valeur des champs affectant la façon dont les paquets peuvent être traités ainsi que la taille du *payload* ;
- une version Java expérimentale utilisant un *Message Broker* pour répartir l'opération de signature sur plusieurs serveurs.

NDNperf propose une majorité d'options en rapport avec le processus de signature car nous l'avons identifié comme le principal *bottleneck* dans les applications. Effectivement, le profilage du code en utilisant Valgrind sur un serveur NDN a montré que la majorité du temps processeur

10. `https://www.twitch.tv/`
11. `https://www.pscp.tv/`

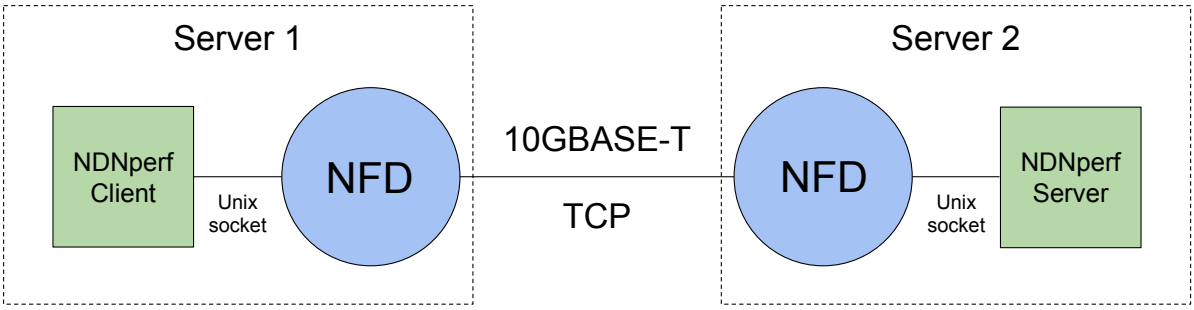


FIGURE 3.1 – Implémentation du réseau NDN minimaliste sur notre Testbed

était utilisée pour la signature des paquets *Data* telle que prévue dans le fonctionnement des ICN (contenus vs connexions). Entre 87% et 64% du temps CPU sert ainsi au chiffrement pour respectivement 1024 et 8192 octets de *payload*. Ainsi, il paraît judicieux d'essayer d'améliorer l'efficacité de cette étape. La deuxième étape la plus coûteuse est l'encodage des paquets, représentant 6% pour un paquet avec un *payload* de 1024 octets et jusqu'à 30% avec un de 8192 octets.

Pour nos expériences, nous avons utilisé 2 serveurs DELL PowerEdge R730. Chacun contient deux processeurs Intel Xeon à 8 cœurs avec une fréquence de fonctionnement de 2.4GHz (E5-2630v3) et les technologies HyperThreading et Turbo Boost activées. Les serveurs possèdent 128GB de RAM ainsi que deux SSD SAS 400GB montés en RAID0 pour le système d'exploitation. Les serveurs sont interconnectés directement entre eux (ad hoc) à une vitesse de 10 gigabits grâce à deux cartes réseau Intel X540. Les serveurs utilisent les versions 0.4.0 de la librairie NDN et du daemon NFD et notre configuration réseau est donnée en Figure 3.1.

La Figure 3.2 nous donne un aperçu des débits atteignables par nos trois implémentations d'NDNperf (deux Java et une C++) avec des paquets *Data* qui contiennent 8192 octets de données, une signature RSA-2048 et en faisant varier la fenêtre d'émission. À cause d'une latence plus importante et le fait que le calcul de la signature prenne plus de temps pour les applications Java, ces implémentations sont plus lentes que leur équivalent C++. La version utilisant le *Message Broker* (RabbitMQ) obtient de meilleurs résultats avec des fenêtres d'émission plus importantes (débits jusqu'à 75% plus important) mais au prix d'un serveur supplémentaire dédié à la signature des paquets. Au vu de ces résultats, nous ne considérerons que la version C++ de NDNperf dans nos prochaines expériences.

Avant chaque test, nous faisons une session d'échauffement en renouvelant plusieurs fois le cache des nœuds NFD car sinon, le débit a tendance à osciller pendant quelques temps et nous voulons éviter de possibles erreurs de mesures. Dans les prochaines expériences, nous utiliserons un MTU de 1500 et une fenêtre de 8 paquets *Interest* pour le client car une expérience préliminaire montre que c'est la valeur idéale (sweet spot). En effet, l'expérience illustrée en Figure 3.3 relève le débit en paquets par seconde pour deux tailles de paquets et deux types de signatures (un hash SHA-256 ou une signature RSA-2048) et ce, pour différentes tailles de fenêtre d'émission pour un serveur single-thread. Quand la Signature est RSA-2048, le débit maximum est atteint à partir de fenêtres de 4 paquets ou plus. Mais dans le cas d'un hash SHA-256, il y a un renversement de tendance entre petits (1024 octets) et gros paquets (8192 octets) à partir d'une fenêtre de 8 paquets. Pour cela, nous utiliserons une fenêtre de 8 paquets qui représente le meilleurs compromis. Lorsque nous utiliserons un serveur multi-threads, nous multiplierons cette valeur par le nombre de threads alloués à la signature des paquets.

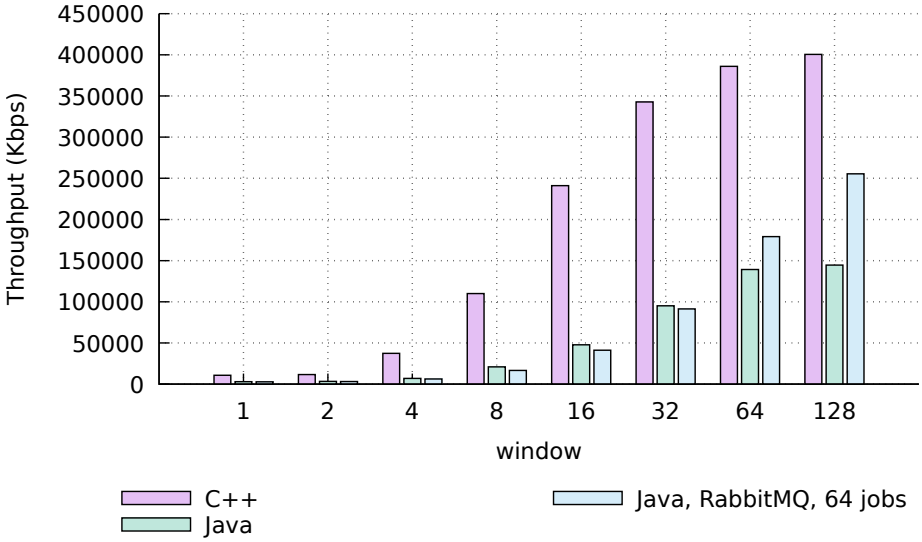


FIGURE 3.2 – Débit atteignable par les 3 différentes implémentations de NDNperf lors de la génération de nouveaux paquets *Data*

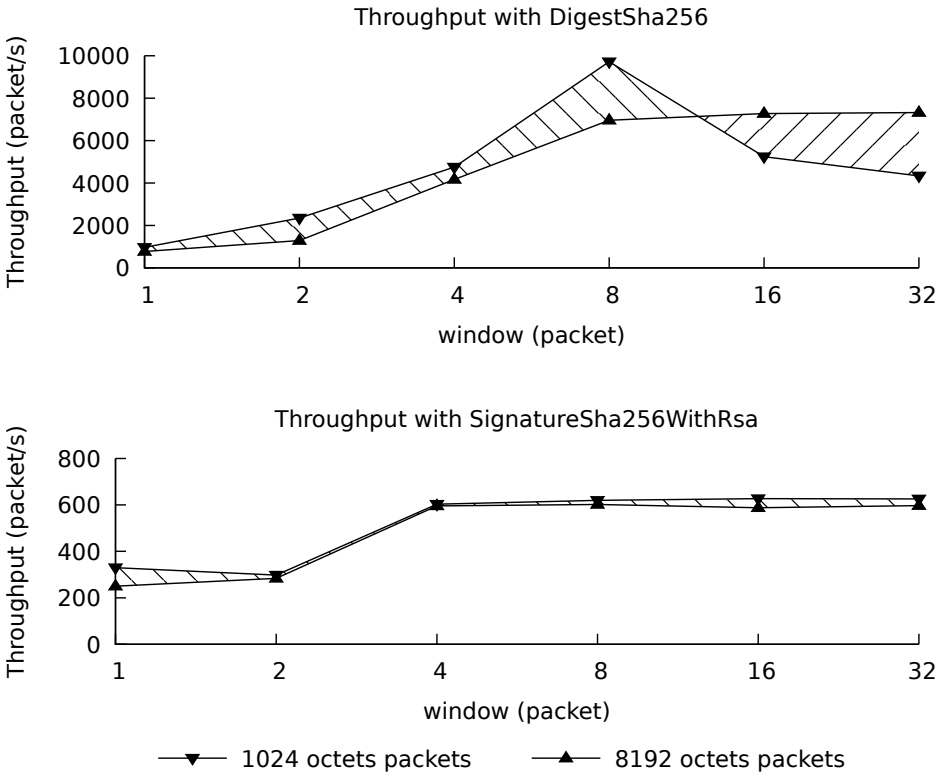


FIGURE 3.3 – Nombre de nouveaux paquets par seconde émis selon la taille de la fenêtre et différents paramètres (taille de paquets et signature)

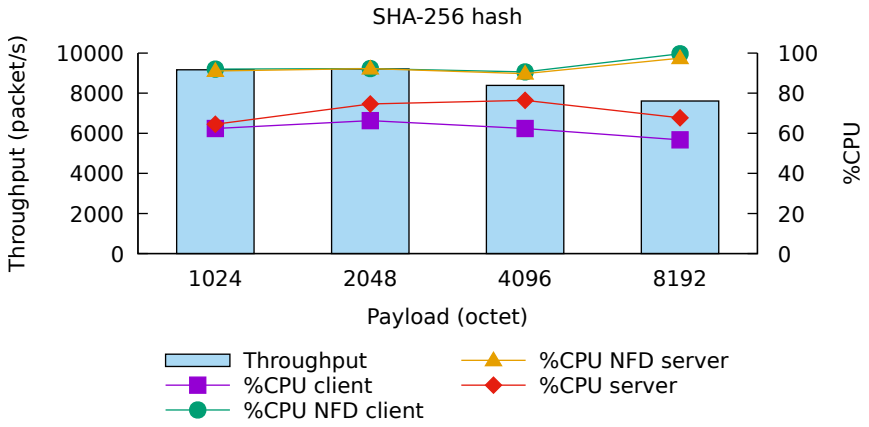


FIGURE 3.4 – Débits de nouveaux paquets *Data* et utilisations CPU associées pour *DigestSha256*

3.3 Évaluation

3.3.1 Évaluation d'un fournisseur de contenus single-thread

Notre première évaluation est basée sur une version single-thread de NDNpef. Le débit atteignable est testé dans ces quatre conditions : en générant de nouveau paquets *Data* avec (1) *DigestSha256* ou (2) *SignatureSha256WithRsa*, et en cherchant des contenus déjà présents dans (3) le cache du nœud NFD près de l'utilisateur ou (4) celui proche du fournisseur de contenus. Lors de ces expériences, nous donnerons les résultats pour quatre tailles de paquet (*payload* de 1024, 2048, 4096 ou 8192 octets). Pour les deux premières expériences, nous utilisons les paramètres par défaut d'NFD ainsi qu'une fraîcheur de 0 ms pour les paquets *Data* pour que le client ne les récupère pas d'un des caches des nœuds du réseau. La Figure 3.4 nous montre les débits observés en paquets par seconde et l'utilisation du processeur (1 cœur) de nos quatre applications (client NDNperf, NFD côté client, NFN côté serveur et NDNperf serveur). On peut voir que le nombre de paquets qu'est capable de transmettre NFD diminue avec la taille de ceux-ci à partir de 4096 octets de *payload* (environ 10% à chaque fois) mais cela est très largement compensé par le fait que l'on double la quantité d'information contenue dans ces paquets. Ainsi globalement, le débit augmente avec la quantité d'information contenue dans chaque paquet avec un maximum atteint de 487 Mbps lorsque le *payload* est de 8192 octets. Dans le cas où l'on utilise qu'un simple SHA-256 comme Signature de paquet, une application single-thread est suffisante pour saturer un nœud NFD car, comme le montre notre expérience, notre serveur NDNperf utilise au plus 80% d'un cœur de nos processeurs alors que les deux nœuds NFD forwardant des paquets sont saturés (CPU >95%).

Mais lorsque l'on passe à un algorithme permettant de réellement signer ses paquets (SHA-256 n'étant qu'un digest mais NDN le place quand même dans le champ « Signature ») le *bottleneck* change. Ainsi, comme le montre la Figure 3.5, notre serveur NDNperf devient l'élément limitant de notre réseau lorsque celui utilise une clé RSA-2048 (taille par défaut). Signer des paquets avec une clé RSA de 2048 bits est une opération coûteuse et une application single-thread n'est pas adaptée pour ce type de signature car, comme le montre notre expérience, le serveur NDNperf n'est capable de générer qu'au plus 550 paquets et le débit maximum atteignable est d'environ 34 Mbps avec un *payload* de 8192 octets. Dans ces conditions, le serveur NDNperf est de loin l'élément limitant du réseau, les nœuds NFD n'utilisant que 21% d'un cœur de processeur. Ce

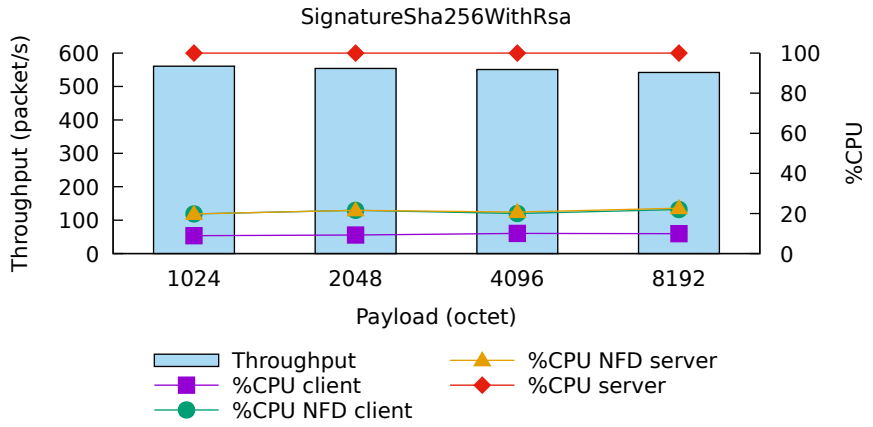


FIGURE 3.5 – Débits de nouveaux paquets *Data* et utilisations CPU associées pour *Sha256WithRsa*

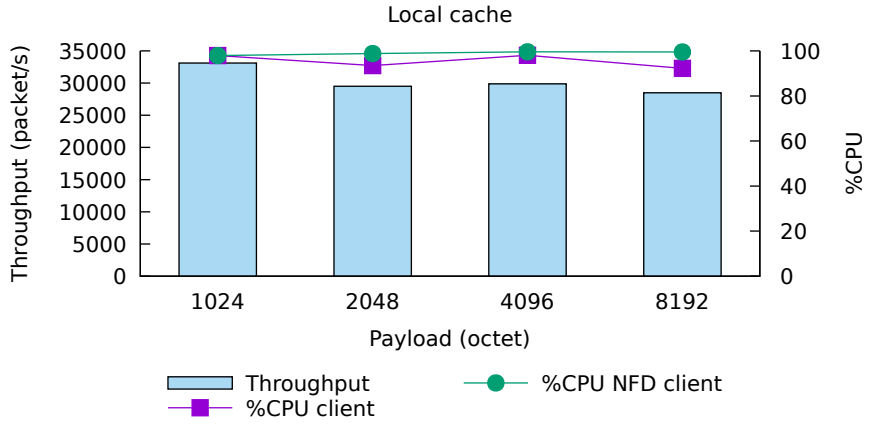


FIGURE 3.6 – Débits du cache du nœud NFD côté client et utilisations CPU associées

constat montre la limite actuelle de NDN lorsque les données fraîches doivent être envoyées (streaming live, etc.).

3.3.2 Évaluation de NFD

Les deux prochaines expériences cherchent à mesurer le débit que peuvent fournir les nœuds NFD lorsque qu'ils font appel à leur cache local ou celui d'un autre nœud que l'on nommera cache distant. Pour ces expériences, nous avons augmenté la capacité des caches NFD à 2^{21} paquets (défini par défaut à 2^{16}) pour que l'expérience soit suffisamment longue pour avoir une mesure fiable, mais cela baisse cependant un peu le débit maximum des nœuds car il faudra un peu plus de temps pour effectuer un *lookup* dans le cache. Pour chaque taille de paquet, nous remplissons le cache des nœuds NFD de paquets préalablement signés avec la fonction *SignatureSha256WithRsa*.

La figure 3.6 montre le débit en paquets par seconde et l'utilisation du processeur pour le client NDNperf et son nœud NFD. En comparaison avec les expériences précédentes, le débit atteint est très fortement supérieur à ce que l'on avait mesuré avec près de trois fois le débit

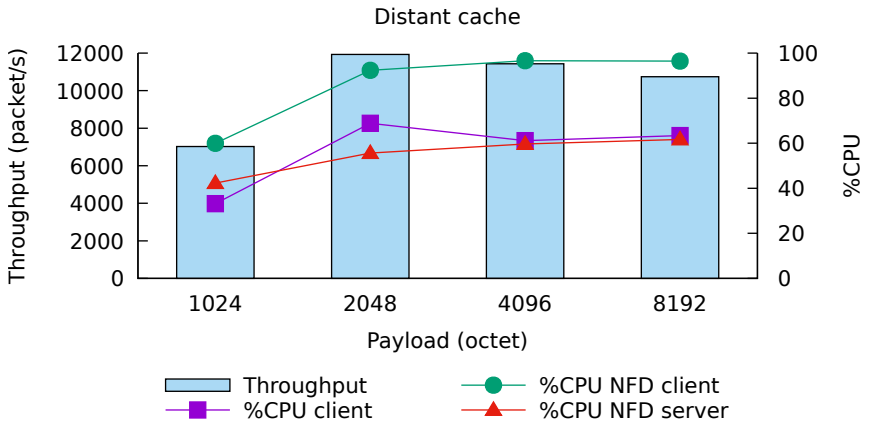


FIGURE 3.7 – Débits du cache du nœud NFD côté serveur et utilisations CPU associées

atteint durant l'expérience avec les paquets contenant un digest SHA-256 (Figure 3.4). Avec un *payload* de 8192 octets, le débit relevé est d'environ 1792 Mbps. Cette grande différence peut s'expliquer car le nœud NFD n'a besoin que de regarder dans son cache pour récupérer le paquet correspondant à la requête du client et n'a donc pas besoin de réaliser les opérations suivantes nécessaires pour transmettre le paquet *Interest* du client au prochain nœud, comme la consultation de la FIB. Cette expérience montre aussi que le protocole NDN est efficace lorsque des paquets sont réutilisés des caches.

Ensuite, La figure 3.7 nous donne le débit en paquets par seconde lorsque que le paquet demandé n'est pas disponible dans le cache proche du client mais dans celui du nœud suivant. Ainsi le premier nœud NFD devra effectuer toute la chaîne d'opérations pour transmettre la requête du client. Dans ces conditions, le débit maximum atteignable chute drastiquement à 671 Mbps avec un *payload* de 8192 octets (contre 1792 Mbps), débit que l'on peut considérer décevant mais celui-ci est tout de même supérieur au débit relevé lors de la génération de nouveaux paquets (487 Mbps avec SHA-256). Comme pour la génération de paquets avec SHA-256, on peut observer une légère baisse du débit en paquets par seconde avec l'augmentation de la quantité d'information contenue dans les paquets (environ -5%). Le débit relevé pour un *payload* de 1024 octets est surprenamment faible mais il n'est pas dû à une erreur de mesure car reproductible. Pour des paquets plus volumineux, le nœud NFD client est clairement l'élément limitant et le second nœud n'utilise que 60% d'un cœur processeur. On peut conclure que les opérations nécessaires au routage des paquets sont extrêmement coûteuses et représentent un tiers du coût total des opérations effectuées par le router. Ces résultats confirment la chaîne opérationnelle trop complexe reprochée à NFD [YSC12].

3.3.3 Évaluation d'une version multithread d'un fournisseur de contenus

Pour notre prochaine expérience, nous utilisons notre serveur NDNperf dans sa version multi-threads. Nous comparons les gains en débit pour les paquets signés par rapport à la version single-thread de la Figure 3.5. Pour simplifier la lecture des prochaines figures, nous utiliserons seulement des paquets avec un *payload* de 8192 octets sauf mention contraire. La Figure 3.8 montre ainsi le débit pour différents nombres de thread utilisés pour la signature. Lors de cette expérience nous faisons varier le nombre de threads de 2 à 32 par puissance de 2, 32 étant la valeur où l'on arrive à saturer les nœuds NFD. Il faut savoir que le débit obtenu avec 32

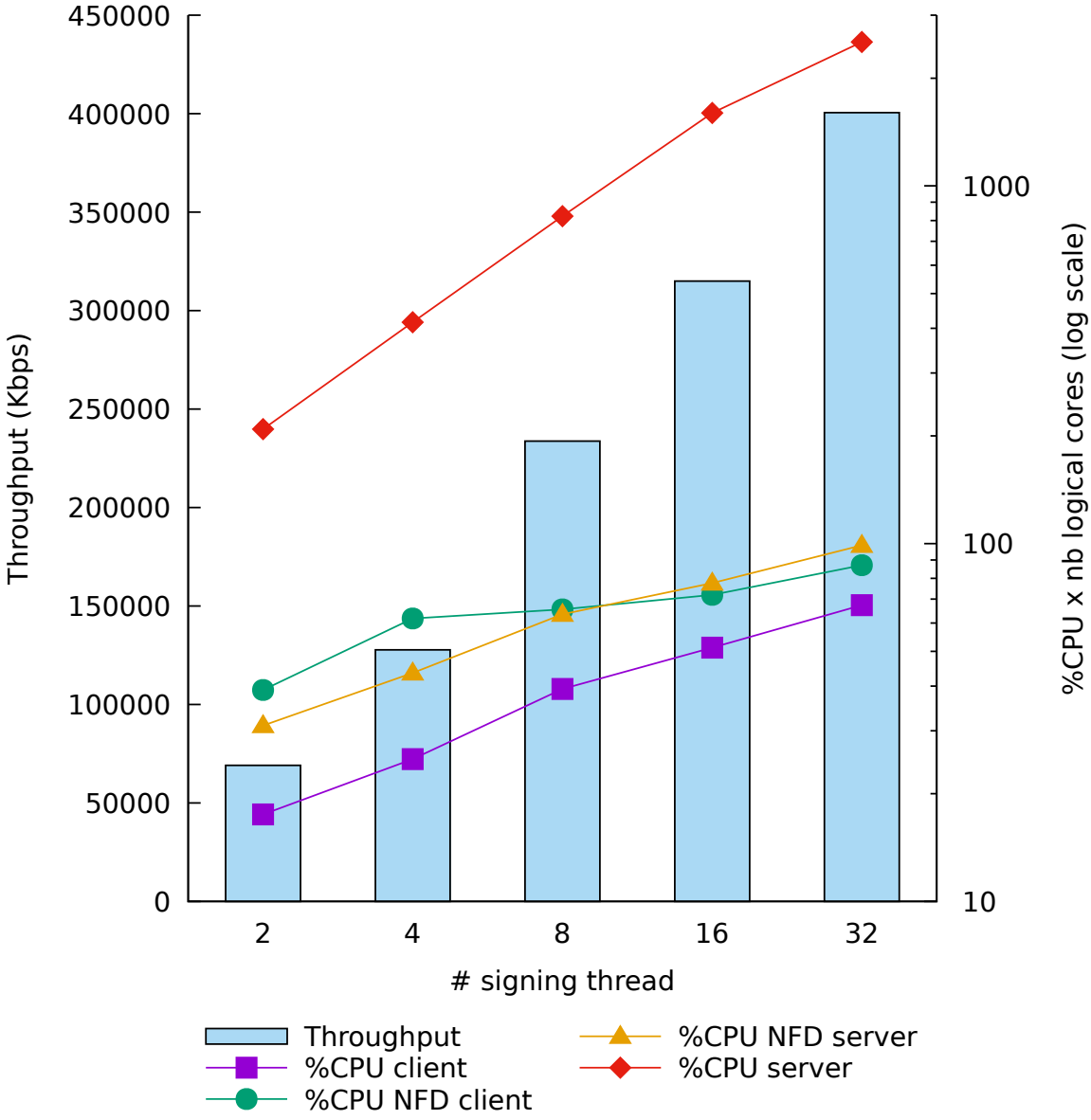


FIGURE 3.8 – Débits de nouveaux paquets *Data* et utilisations CPU associées pour *Sha256WithRsa* (multithread)

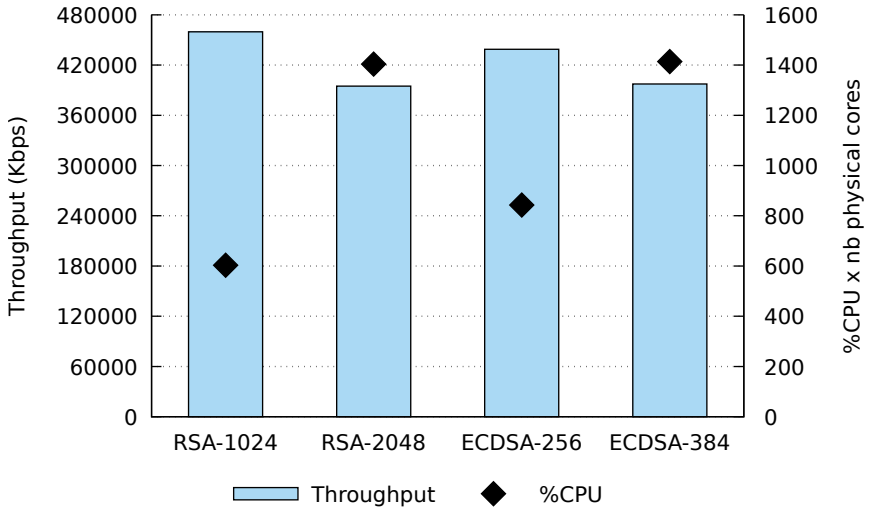


FIGURE 3.9 – Débits de nouveaux paquets *Data* et utilisations CPU associées avec les signatures disponibles dans la librairie

threads peut quasiment être atteint avec 16 threads lorsque l'hyper-threading est désactivé car il n'accélère que très peu l'exécution du programme. C'est d'ailleurs pour cela que l'augmentation du débit ralentit à partir de 16 threads sur notre figure, début de l'utilisation des cœurs logiques. Lorsque le serveur utilise 32 threads, on peut relever que « seulement » 25 sont réellement utiles mais utiliser presque toutes les ressources processeur d'un serveur relativement correct pour la génération de paquets semble toujours inefficace et un verrous important au déploiement de NDN. Au final, l'utilisation de 32 threads montre un débit 11.5 fois supérieur à celui de notre serveur single-thread, atteignant 401 Mbps.

En conclusion, NFD est le *bottleneck* de notre *testbed* lors de transfert de paquets à partir du cache des nœuds du réseau ou lorsque un serveur génère des paquets avec SHA-256. En revanche, avec une signature comme RSA-2048, c'est le serveur NDNperf qui devient *bottleneck* à cause du coût très élevé de la signature. Avec du matériel performant générant les paquets, il est possible d'utiliser au maximum les capacités du réseau mais au prix de l'utilisation quasi complète des ressources du serveur. Dans le cas d'applications temps réel, cela peut être un problème d'allouer une machine complète pour générer un flux réseau si faible (en excluant des cas comme la diffusion de chaînes télévisées ou ce compromis peut être acceptable à cause du nombre très important de spectateurs).

3.4 Réduire le coût de la signature

3.4.1 En changeant l'algorithme de signature

La librairie NDN propose actuellement 2 types de signatures, à savoir les algorithmes RSA et ECDSA, mais quelques indices dans le code source montrent d'éventuels algorithmes comme HMAC et AES. Avec la librairie C++, on ne peut que générer deux tailles pour chacune des deux clés RSA (1024, 2048) et ECDSA (256, 384). La Figure 3.9 montre les débits atteignables pour la génération de nouveaux paquets avec ces différentes signatures. Comme certaines signatures sont moins coûteuses que d'autres, nous désactivons l'HyperThreading pour cette expérience pour pouvoir faire une comparaison équitable.

TABLE 3.1 – Débits et charge du serveur dans différents scénarios

Source du paquet *Data*	Débit	# cœurs serveur
Cache côté client	1792 Mbps	0
Cache côté serveur	671 Mbps	0
Serveur avec SHA256	487 Mbps	1
Serveur avec RSA-1024	460 Mbps	6
Serveur avec RSA-2048	394 Mbps	14
Serveur avec ECDSA-256	439 Mbps	8
Serveur avec ECDSA-384	397 Mbps	14

Pour RSA, réduire la taille de la clé réduit fortement les ressources nécessaires pour la signature des paquets et seuls 6 threads sont nécessaires au lieu de 14 pour atteindre le débit maximum. Bien sûr, réduire la taille de la clé réduit aussi le niveau de sécurité que celle-ci procure, mais cela peut être un compromis acceptable pour certaines applications peu critiques. Pour ECDSA, avec une clé de 256 bits, le débit est similaire à RSA-1024 mais offre un niveau de sécurité similaire à RSA-3072 tout en ayant besoin de « seulement » 8 threads de signature. Par contre, cela se fera au détriment du client qui va devoir allouer plus de ressources pour vérifier cette signature qu'avec RSA-2048 (transferts de données moins efficient énergiquement). Ainsi, dans le cas d'applications qui peuvent tirer parti du cache du réseau ou qui peuvent générer à l'avance leur paquets, ECDSA n'est pas forcément le meilleur choix car elles vont réduire les performances de leur clients sans nécessairement voir de gain de leur côté. Pour finir, ECDSA-386 offre les mêmes performances que RSA-2048 avec un niveau de sécurité largement supérieur mais risque aussi d'impacter fortement le client. Au vu des résultats, pour une application temps réel nous recommandons donc d'utiliser ECDSA-256 car il permet d'offrir le meilleur compromis entre sécurité et utilisation du processeur. Les résultats obtenus pour un *payload* de 8192 octets sont synthétisés dans le Tableau 3.1.

3.4.2 En améliorant la fonction de signature de NDN

Chaque fois que l'on demande la signature d'un paquet, la librairie NDN lit le fichier qui contient la clé privée, ce qui n'est pas efficace. De plus, les noms des fichiers des clés publiques et privées sont des hashs du nom NDN de cette paire de clés. La librairie NDN a donc besoin de calculer à chaque fois deux fois le hash (un pour chaque clé) pour pouvoir calculer la signature. Toutes ces opérations non-essentielles ralentissent inutilement la génération de signature et pourraient être facilement évitées. L'idée est d'autoriser le fournisseur de contenus à stocker dans sa mémoire la clé qu'il veut utiliser et les informations associées comme le type de signature dans une structure dédiée que l'on pourrait transmettre à la fonction de signature. La Figure 3.10 montre que le temps nécessaire pour signer un paquet est grandement réduit (en moyenne 250 microsecondes par appels de la fonction) avec notre méthode de signature personnalisée, basée sur une structure en mémoire, comparé à celle fournie par la librairie NDN pour les deux signatures RSA et ECDSA. Cette amélioration a un effet positif sur le débit des serveurs lorsque ceux-ci sont *bottleneck* comme illustré par la Figure 3.11. L'expérience a été réalisée avec 4 threads de signature, un *payload* de 8192 octets et pour les deux signatures RSA-2048 et ECDSA-256. Dans ces conditions, notre fonction de signature augmente le débit de 13% pour RSA et de 20% pour

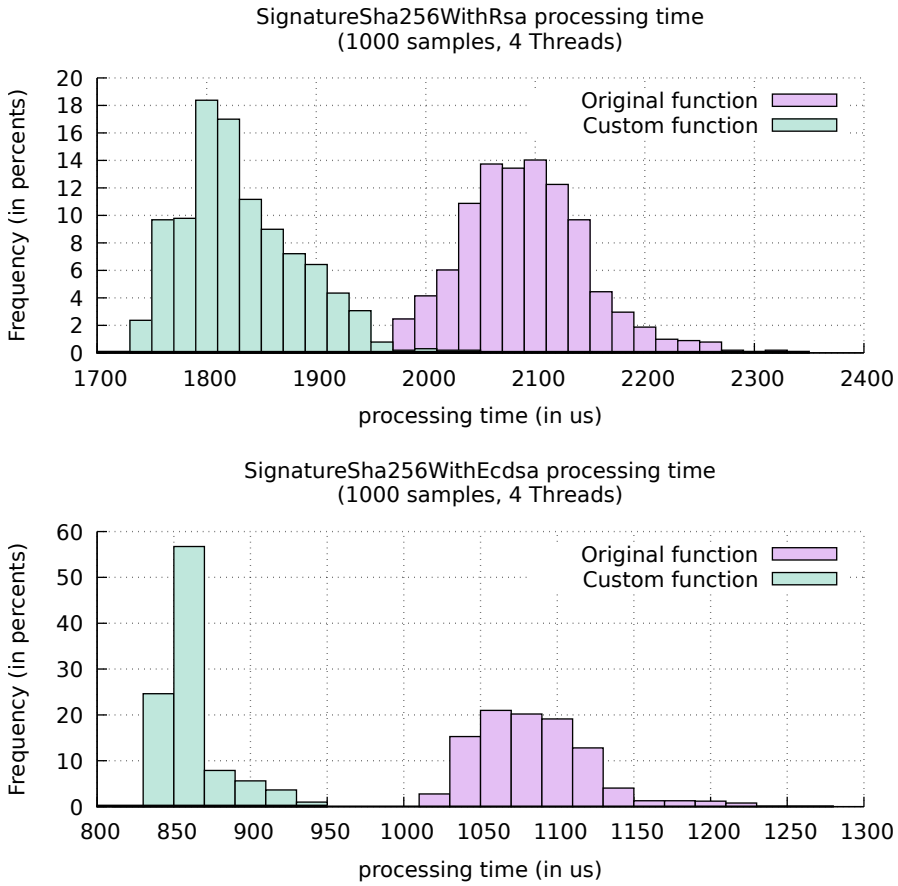


FIGURE 3.10 – distribution du temps nécessaire à la signature de paquets *Data* pour les fonctions de signature originelle et optimisée (1 sample = moyenne sur 2 secondes)

ECDSA.

3.4.3 En réduisant le nombre de signatures nécessaires pour la transmission de contenus

Une autre façon de réduire le coût de la signature lors de transfert de contenus est de réduire le nombre de paquets nécessaires pour les transmettre. Il existe plusieurs méthodes pour réduire le nombre de signatures nécessaires pour un contenu, on peut par exemple citer la méthode utilisant un *manifest* et qui est déjà utilisée dans les transferts de fichiers actuels, par exemple pour mettre à jour des logiciels. Avec le protocole NDN, il est possible de créer des paquets qui contiendront les hashs de plusieurs paquets et ainsi une seule signature sera nécessaire pour tous ces hashs. Cependant, cette méthode requiert une logique supplémentaire pour récupérer ces *manifests* car ils devront être dissociés du contenu. De plus, elle n'est pas adaptée pour les applications sensibles au délai car elle induit un délai supplémentaire le temps de temporiser un certain nombre de paquets pour produire un *manifest*. Une autre méthode, reprenant le principe des *blockchains*, peut être mise en place. Le principe serait de ne signer que le dernier paquet et chaque paquet ferait référence au paquet précédent en contenant son hash. Cette méthode ne nécessiterait qu'une seule signature par contenu mais celui-ci doit être complet pour pouvoir être validé, ce qui est, encore une fois, difficilement applicable dans une application sensible au délai.

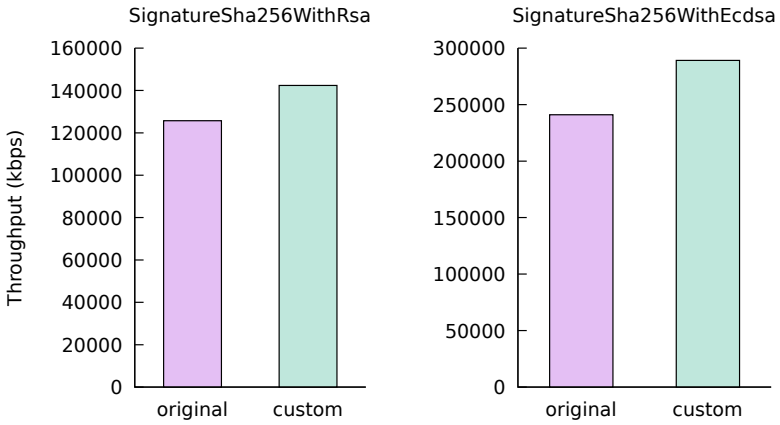


FIGURE 3.11 – Comparaison des débits du serveur avec les deux fonctions de signature (originelle et optimisée)

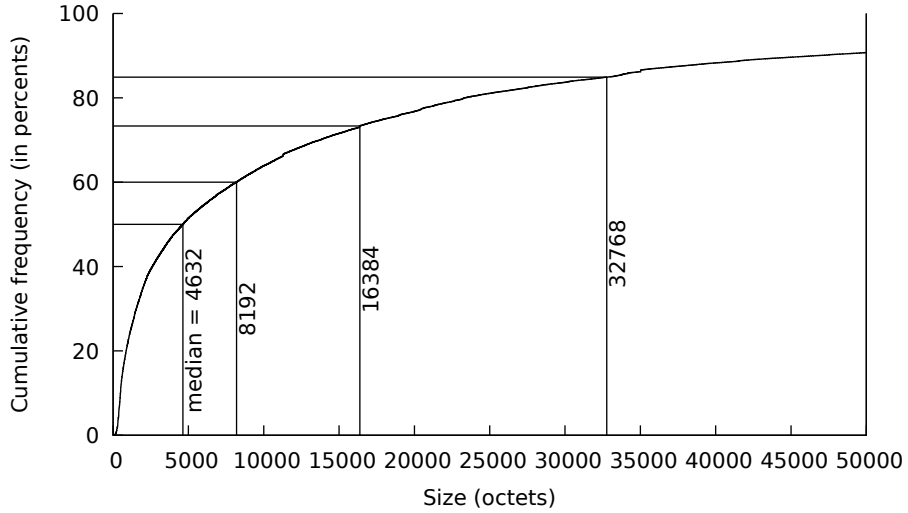


FIGURE 3.12 – Fréquence cumulée des tailles des réponses à des requêtes HTTP GET et dont le code de retour est 200 (1,7 million)

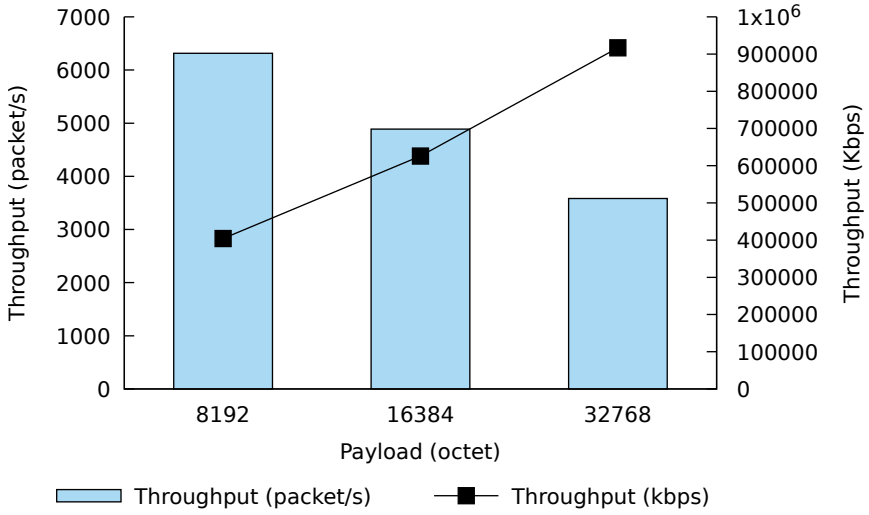


FIGURE 3.13 – Débits de nouveaux paquets *Data* signés avec *Sha256WithRsa* pour différentes tailles de *payload*

Ici, nous nous limiterons à une méthode simple, ne demandant pas de traitements supplémentaires pour être appliquée, ainsi nous allons juste augmenter la quantité de données contenue dans chaque paquet, ce qui permettra de réduire le nombre total de signatures nécessaire pour transmettre un contenu. Actuellement, NDN limite la taille maximale de ses paquets à 8800 octets mais ce paramètre peut être modifié dans le code source. Si nous comparons cette limite avec les tailles des contenus que l'on peut trouver sur Internet [12] (Figure 3.12), cette taille permet de transmettre environ 60% de ces contenus avec un seul et unique paquet. Ainsi en modifiant la taille maximale des paquets NDN, les contenus plus volumineux comme les vidéos nécessiterons moins de paquets pour être transmis et donc moins de signatures. Comme la Figure 3.13 le montre, augmenter la taille maximale des paquets NDN a aussi un effet positif sur le débit que peut transmettre le réseau même si cela réduit la quantité de paquets que les nœuds NDN sont capables de traiter par seconde, le débit maximum atteint est de 917 Mbps avec des paquets de 32786 octets. La capacité des applications à bénéficier de cette amélioration dépendra de leur manière de communiquer et de leur tolérance à la latence. Cette méthode va aussi augmenter la fragmentation des paquets NDN (soit réalisée par NDN lui-même, soit par IP en cas d'overlay), ce qui peut se traduire par un plus haut taux d'erreur de transmission. Néanmoins, la fragmentation est bien maîtrisée et la présence de cache sur chaque nœud du réseau et de mécanismes de (re)transmission intelligents, par exemple avec NDNLPv2, limiteront les problèmes en cas de perte de fragments.

Pour finir, la Figure 3.14 compare l'efficacité des paramètres par défaut de NDN aux 3 optimisations que nous avons énumérées précédemment (algorithme de signature, clé en mémoire, taille des paquets). Pour un coût processeur donné, nos optimisations augmentent grandement l'efficacité de notre serveur comparé aux paramètres par défaut. Par exemple, avec 4 cœurs processeur, nous obtenons environ 125 Mbps avec les paramètres par défaut alors qu'il est possible d'obtenir environ 800 Mbps avec les optimisations proposées, soit un débit 6,4 fois supérieur.

12. Données venant de httpparchive.org, mars 15 2016

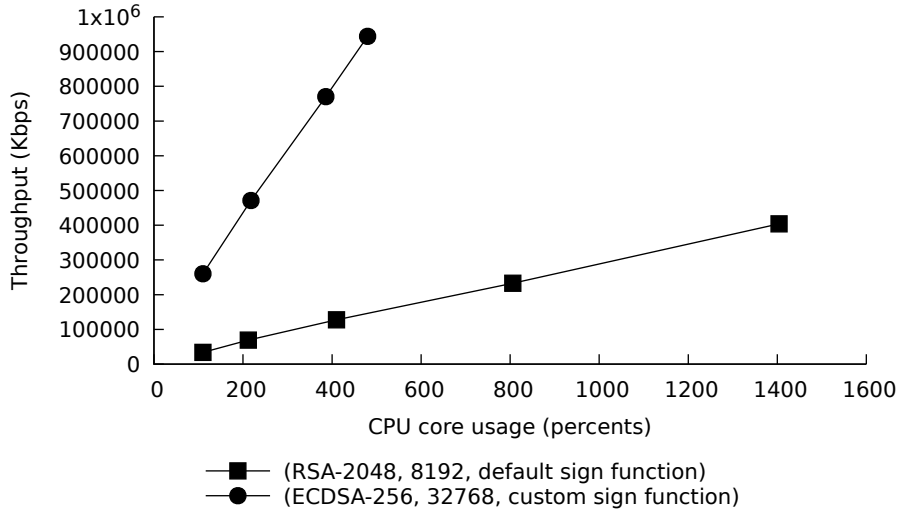


FIGURE 3.14 – Débits de nouveaux paquets *Data* en fonction du nombre de threads pour différentes configurations de signature

3.5 Conclusion

Nous avons conduit une évaluation rigoureuse des performances de NDN côté serveur. Nous avons développé NDNperf, un outil de mesure des performances maximales atteignables par une application NDN. Nos expériences montrent qu'il n'est pas nécessaire d'avoir un client multithread pour pouvoir profiter des capacités du réseau. Par contre pour ce qui est des serveurs NDN, la situation est différente. Si l'on exclu le cas particulier où l'authentification des paquets n'est pas nécessaire (*DigestSha256*), calculer une signature (pour chaque paquet) à l'aide d'algorithmes de chiffrement asymétriques nécessite beaucoup de ressources processeur et un serveur multithread semble nécessaire pour atteindre un débit décent. Cependant, avec les paramètres par défaut, un serveur classique sur architecture x86 semble encore insuffisant pour qu'un fournisseur de contenus ne soit pas le *bottleneck* du réseau, car NDNperf ne réalise de surcroît que très peu d'opérations entre la réception et l'envoi de paquets. Mais nous avons aussi montré qu'il est possible d'utiliser l'algorithme ECDSA, qui permet une signature efficace au prix d'une vérification plus complexe par les clients. Cette signature peut être intéressante pour les applications temps réel ou qui ne peuvent pas profiter pleinement du cache des nœuds du réseau. Nous avons aussi évalué le gain de certaines optimisations qui consistent à éviter des opérations redondantes sur la manipulation des clés en mémoire, ou encore l'augmentation de la taille maximale des paquets. Celles-ci améliorent significativement le débit par thread de notre application allant jusqu'à une accélération de 6.4 une fois combinées. Des techniques plus avancées comme la signature assistée par GPU pourraient encore accélérer cette étape qui reste la plus coûteuse du protocole.

Cependant, le coût de génération des paquets n'est pas le seul problème pouvant limiter les performances d'un réseau NDN. En effet, plusieurs types d'attaques ciblant les différentes tables d'un routeur NDN peuvent causer des dénis de service importants, comme indiqué dans l'état de l'art. Dans le prochain chapitre, nous nous intéressons ainsi à l'attaque par empoisonnement de contenus (CPA).

Chapitre 4

Étude de la sécurité du protocole NDN : cas de l'attaque par empoisonnement de contenus

Sommaire

A noter : ce travail a été réalisé en collaboration avec l'Université Technologique de Troyes (UTT), notamment avec les personnes suivantes : Tan Nguyen, Guillaume Doyen et Rémi Cogranne [NMD+17].

4.1 Introduction

La sécurité des protocoles ICN, ainsi que leur mise en œuvre doit d'abord être évaluée pour en faire des alternatives sûres qui pourraient être facilement adoptées par de potentiels intéressés. Alors que la plupart des efforts de recherche se sont concentrés sur les performances de mise en cache ou sur des attaques très spécifiques comme l'attaque par inondation d'*Interests* (IFA), moins d'attention a été portée sur les attaques visant le cache.

Plus précisément, l'attaque par empoisonnement de contenus (CPA) est identifiée par le comité d'NDN comme la deuxième menace la plus importante juste après l'IFA [13], bien qu'elle n'ait pas encore fait l'objet d'une étude approfondie. Cependant, l'une des questions les plus importantes de la CPA repose sur l'absence d'une mise en œuvre détaillée, ainsi que d'une étude spécifique et complète du phénomène dans un scénario se rapprochant de la réalité. Cela empêche les chercheurs d'acquérir des connaissances sur la faisabilité et l'impact de la CPA et, bien sûr, de concevoir des solutions qui pourraient améliorer la résilience du protocole face à cette attaque.

Afin de combler ces lacunes, nous proposons une description détaillée de la CPA lors de déploiements réalistes et procédons à une caractérisation détaillée de cette attaque. Nous présentons les résultats d'une campagne de mesure réalisée grâce à une méthodologie rigoureuse où (1) nous définissons trois scénarios d'attaque réalistes de CPA malgré les récents mécanismes de protection ; (2) nous les mettons en œuvre sur un banc d'essai et mesurons l'effet des paramètres d'attaque à travers de nombreuses expériences ; (3) nous évaluons l'impact de l'attaque sur les principaux acteurs du réseau (utilisateurs, routeurs d'accès et de cœur de réseau, fournisseurs de contenus) ; et (4) nous proposons une solution de prévention et des pistes pour d'éventuels détecteurs.

Le reste du chapitre est organisé comme suit. La section 4.2 décrit les scénarios d'attaque qui font l'objet d'une étude approfondie dans la section 4.3 par le biais d'expériences visant à évaluer et à caractériser leur impact sur le réseau. Enfin, la section 4.4 décrit deux solutions réduisant l'impact de l'attaque : l'une basée sur un patch logiciel corrigeant une vulnérabilité, et l'autre sur des contre-mesures dynamiques. La section 4.5 conclut sur la menace que constitue la CPA à l'aune de ces travaux.

4.2 Scénarios d'attaque

Dans cette section, nous décrivons la topologie que nous avons déployée pour l'étude de l'impact de la CPA, ainsi que 3 scénarios d'attaque qui peuvent être mis en œuvre pour insérer de mauvais paquets *Data* dans le cache. Ces scénarios sont nommés : (1) *unregistered* remote provider ; (2) *multicast* forwarding and (3) *bestroute* forwarding. Nous ne considérons pas le cas où un routeur serait compromis et qui pourrait facilement répondre à n'importe quel paquets *Interest* avec un mauvais paquet *Data*, car prendre le contrôle d'un élément du réseau est très compliqué dans un vrai contexte opérationnel alors que lancer l'attaque depuis les extrémités du réseau que ce soit un fournisseur de contenus ou un utilisateur pour déployer l'attaque est plus réaliste. Par conséquent, nos trois scénarios considèrent des cas où la CPA est effectuée par un seul fournisseur de contenus non légitime ou par une collaboration entre mauvais fournisseurs de contenus et utilisateurs coordonnés par un unique attaquant.

4.2.1 Topologie et comportements d'attaquants

La topologie utilisée pour l'étude de la CPA est illustrée dans la Figure 4.1. Nous pensons que cette topologie, ainsi que les comportements de tous ces composants, sont suffisamment complets pour atteindre l'objectif principal de prouver la faisabilité et l'impact de la CPA en raison des caractéristiques suivantes :

1. Elle montre le rôle général des différents nœuds/fonctions impliqués dans cette attaque pour NDN et reflète une structure typique d'opérateur de réseau : un routeur central R2, avec un grand cache, où se trouvent les fournisseurs de contenus, un routeur intermédiaire

13. (voir http ://named-data.net/project/faq/)

R1 sur la route de R2 vers le bon fournisseur de contenus, ce qui aide à réduire la latence de ses paquets *Data* populaires et un routeur d'accès R3, avec un cache plus petit, où les bons et mauvais utilisateurs sont connectés. De cette façon, cette topologie nous permet de mettre en évidence l'effet de la CPA pour tous les composants impliqués.

2. Le comportement de l'utilisateur est aussi proche que possible de la réalité. Nous supposons que les utilisateurs émettent toujours des paquets *Interest* pour les dernières versions des paquets *Data* d'un contenu (grâce aux champs *MustBeFresh* et *ChildSelector* du paquet *Interest*). Les utilisateurs légitimes envoient des demandes basées sur la popularité des contenus et n'acceptent les paquets qu'après vérification. Quand ils reçoivent un mauvais *Data*, ils ré-émettent un autre *Interest* en ajoutant le champ *Exclude* pour éviter le mauvais paquets *Data* reçu précédemment. Les routeurs ne vérifient pas les paquets *Data* en raison du coût trop élevé de la vérification de la signature des paquets *Data* [GTU14a] à la vitesse du réseau. Les mauvais utilisateurs demandent aussi des contenus selon leur popularité, mais pour des contenus qu'ils ciblent, ils agissent de manière opposée en excluant les bonnes données, favorisant ainsi la diffusion des mauvaises données.

3. Notre topologie ne compte qu'un seul fournisseur de contenus légitime, qui répond aux paquets *Interest* pour tous les contenus. Ce cas correspond à l'un des pires cas, où la disponibilité des contenus légitimes est limitée à une seule route. Le fournisseur de contenus légitime est situé plus loin de R2 où est connecté le *provider* malveillant, ce qui entraîne un délai supplémentaire pour la livraison d'un contenu. Cependant, le cache sur le routeur intermédiaire R1 aide à réduire ce délai une fois que le paquet *Data* y est stocké. En raison de son enregistrement valide, nous considérons que le chemin vers le fournisseur légitime a toujours un coût inférieur à celui vers le mauvais fournisseur. Le bon fournisseur de contenus mettra automatiquement à jour son paquet *Data* lorsque la période de fraîcheur [14] expire.

Il est nécessaire de mentionner que, dans un réseau visant à optimiser l'efficacité de la diffusion et la disponibilité des contenus, un protocole de routage sécurisé (par exemple NLSR [HAA+13]) est trop restrictif et n'est donc pas pleinement reconnu comme un cas réaliste de déploiement du protocole NDN [DP16]. D'ailleurs, les développeurs du protocole NDN ont déclaré que « la gestion de l'espace de nommage ne fait pas partie de l'architecture NDN ». Il n'est donc pas facile de répondre précisément à la question sur la sécurité de l'enregistrement des préfixes, c'est pourquoi NDN reste toujours exposé à l'enregistrement malveillant. Par conséquent, nous soutenons que les fournisseurs de contenus peuvent publier ouvertement leurs contenus sous certains préfixes enregistrés. Un tel enregistrement peut être simple pour un fournisseur légitime, mais il n'est pas évident pour un attaquant de faire enregistrer plusieurs préfixes de fournisseurs. Par conséquent, notre topologie ne compte qu'un seul mauvais fournisseur qui est situé près du routeur central R2. Ce mauvais fournisseur ne répond qu'aux contenus qu'il choisit d'empoisonner. Pour défier le fournisseur légitime, nous considérons que l'attaquant sélectionne toujours les contenus les plus populaires pour la CPA tout en tenant compte d'une distribution de popularité. De telles connaissances peuvent être facilement acquises, par exemple dans le cas du trafic Web par le biais d'informations accessibles au public sur le classement de popularité des sites Web.

Puisque qu'un mauvais paquet *Data* est inutile après avoir été exclu, le mauvais fournisseur doit mettre à jour son paquet *Data* chaque fois qu'il reçoit un paquet *Interest* excluant le mauvais *Data* actuel, pour maintenir la persistance de l'attaque. Pour augmenter le nombre de victimes qui peuvent recevoir un mauvais paquet *Data* en émettant des requêtes naïves (c'est-à-dire sans

14. voir http ://named-data.net/doc/NDN-TLV/current/data.html

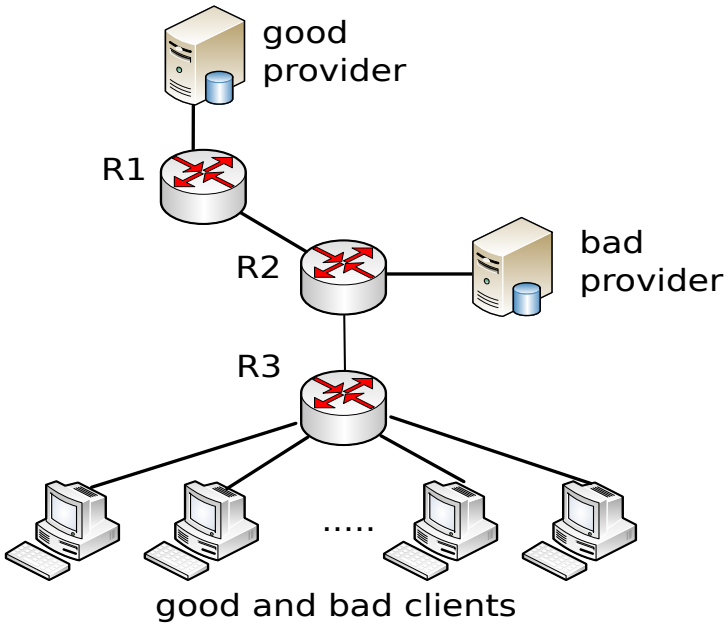


FIGURE 4.1 – Topologie utilisée pour l'étude de la CPA

exclusion des mauvais paquets *Data*), l'attaquant peut définir le champ *FreshnessPeriod* du mauvais paquet *Data* à une valeur élevée.

4.2.2 Scénario *Unregistered remote provider*

Le scénario *Unregistered remote provider* propose d'exploiter une faiblesse que l'on a découvert dans l'implémentation du routeur NDN (NFD) et qui permet un comportement non spécifié au moment de la réception d'un paquet *Data*. Pour chaque paquet *Interest* reçu, le routeur NDN garde en mémoire une trace de cette requête et par quelle(s) *Face(s)* elle à été reçue (in-records) et envoyée (out-records) dans la *Pending Interest Table* (PIT). Néanmoins, ces *out-records* semblent n'être utilisés que lors de la réception de paquets NACK. Cela veut dire que lors de la réception d'un paquet *Data*, quelle que soit la *Face* par laquelle il est arrivé, celui-ci peut satisfaire un paquet *Interest* présent dans la PIT même si ce paquet *Data* ne provient pas d'une *Face* ayant une route pour le nom de ce paquet au préalable (Listing 4.1, ligne 10). Dans ce Listing, on peut voir que le programme commence par ajouter une information relative à la *Face* d'entrée dans l'entête NDNLP du paquet *Data* mais celle-ci ne sera jamais utilisée lors de la recherche dans la PIT. Ensuite, une vérification est faite pour s'assurer que le préfixe « localhost » n'est pas utilisé sur une *Face* qui n'est pas locale. En effet, comme les paquets avec ce préfixe sont le plus souvent des commandes, il est nécessaire de s'assurer que les paquets ne viennent pas de n'importe où. En troisième, nous avons la recherche dans la PIT par la fonction « findAllDataMatches » (Listing 4.2). Celle-ci ne considère pas l'information ajoutée au début de la première fonction puisque seul le nom du paquet est donné en paramètre. La vérification dans la double boucle « for » n'en fera pas plus car celle-ci vérifie que les noms des paquets sont « compatibles » (i.e respecte les contraintes imposées par le paquet *Interest*) et que les champs « Exclude » et « PublisherPublicKeyLocator » sont respectés. Une liste des entrées sera retournée et le paquet *Data* sera utilisé sans plus de vérification si cette liste n'est pas vide puisque que l'on peut voir que l'insertion de ce paquet dans le Content Store suit directement la recherche dans la PIT sans

vérification supplémentaire.

Listing 4.1 – Localisation de la vulnerabilité

```
void Forwarder::onIncomingData(Face& inFace, const Data& data) {
  data.setTag(make_shared<lp::IncomingFaceIdTag>(inFace.getId()));
  ++m_counters.nInData;
  // /localhost scope control
  bool isViolatingLocalhost = inFace.getScope() == ndn::nfd::
      FACE_SCOPE_NON_LOCAL && scope_prefix::LOCALHOST.isPrefixOf(data.getName())
      ;
  if (isViolatingLocalhost) {
    // (drop)
    return;
  }
  // PIT match
  pit::DataMatchResult pitMatches = m_pit.findAllDataMatches(data);
  if(pitMatches.begin()==pitMatches.end()){
    //goto Data unsolicited pipeline
    this->onDataUnsolicited(inFace,data);
    return;
  }
  // CS insert
  m_cs.insert(data);
  {...} //and so on...
}
```

Listing 4.2 – Fonction de recherche dans la PIT

```
DataMatchResult Pit::findAllDataMatches(const Data& data) const {
  auto&& ntMatches = m_nameTree.findAllMatches(data.getName(), &nteHasPitEntries
      );
  DataMatchResult matches;
    for (const name_tree::Entry& nte : ntMatches) {
      for (const shared_ptr<Entry>& pitEntry : nte.getPitEntries()) {
        if (pitEntry->getInterest().matchesData(data))
          matches.emplace_back(pitEntry);
      }
    }
  return matches;
}
```

En exploitant cette faiblesse, un attaquant peut déployer un fournisseur de contenus sur le chemin entre un utilisateur et un fournisseur de contenus légitime sans même avoir besoin d'enregistrer un préfixe (connecté à R2 dans notre topologie). Ce dernier a ensuite seulement besoin d'envoyer des mauvais paquets *Data* pour essayer de faire correspondre des paquets *Interest* en attente sur le routeur. Il existe deux variantes d'attaque utilisant cette faiblesse : en envoyant simplement des paquets *Data* en espérant qu'ils correspondent à une entrée de la PIT (Figure 4.2), ou en répondant à ses propres paquets *Interest* (Figure 4.3). Dans le second cas, l'attaquant ne recevra pas ses paquets *Data* en retour comme un utilisateur normal car une règle dans le routeur NDN empêche le renvoi d'un paquet *Data* par la *Face* par laquelle il est arrivé mais l'insertion dans le cache peut se vérifier avec un nouveau paquet *Interest*. Cette deuxième variante permet aussi d'insérer des paquets *Data* dans le cache même si aucun préfixe n'a été enregistré pour router ce paquet tant que le paquet *Data* arrive avant la suppression de l'entrée PIT correspondante car celle-ci est toujours maintenue un court instant avant sa suppression. Ainsi,

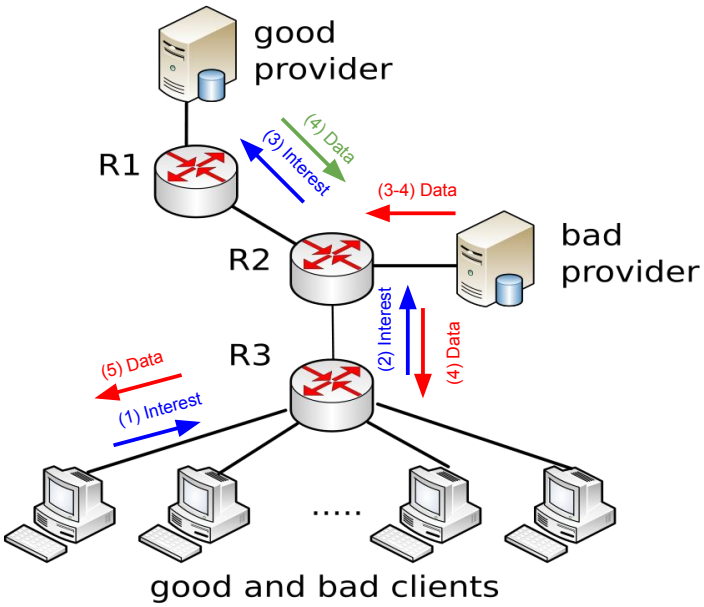


FIGURE 4.2 – Attaque en envoyant des paquets *Data* de façon opportuniste

il est possible de maintenir une entrée dans la PIT en émettant suffisamment fréquemment des paquets Data correspondants (environ 20/s). Dans les expériences de ce chapitre, nous utiliserons la première variante car elle se rapproche plus du comportement d'un fournisseur de contenus.

Il faut savoir que le mauvais fournisseur de contenus est aveugle sur le réseau car il ne reçoit pas les paquets *Interest* des utilisateurs mais peut savoir quel contenus ont été récemment demandés grâce à une attaque d'analyse temporelle [Lau10]. Quand un paquet *Interest* est reçu par R2, une course commence (*race condition*) entre le bon et le mauvais fournisseur de contenus. Seul le premier paquet *Data* qui correspond à ce paquet *Interest* consommera l'entrée dans la PIT et sera transmis (au niveau de R2) alors que les suivants seront ignorés tant qu'une nouvelle entrée n'aura pas été recréée. Par conséquent, un mauvais paquet *Data* a plus de chance de correspondre à un paquet *Interest* pour un contenu spécifique si celui-ci arrive durant la fenêtre de temps $[t_{receive}; t_{receive} + t_{gpDelay}]$ où $t_{receive}$ est le temps où R2 reçoit le paquet *Interest* et $t_{gpDelay}$ est le délai pour recevoir le paquet *Data* correspondant du fournisseur de contenus légitime. Comme il est difficile pour le mauvais fournisseur de contenus d'estimer cette fenêtre, celui-ci doit envoyer ses mauvais paquets *Data* pour des contenus ciblés à un rythme régulier et suffisant pour maximiser ses chances de succès.

4.2.3 Scénario *Multicast*

Multicast est l'une des stratégie de routage possible et actuellement intégrée dans l'implémentation du routeur NDN [ASZ+14]. Lorsque qu'un routeur utilisant cette stratégie reçoit un paquet *Interest*, il le transmet à toutes les *Faces* qui ont enregistré un préfixe correspondant dans la FIB du routeur. Alors que la CPA était réalisée par le seul effort d'un mauvais fournisseur de contenus dans le cas du scénario précédent d'*unregistered remote provider*, le scénario *multicast* a besoin quant à lui d'une collaboration avec des utilisateurs malveillants dans le but de demander et d'insérer de mauvais paquets *Data* dans le cache des routeurs. Plus spécifiquement, les mau-

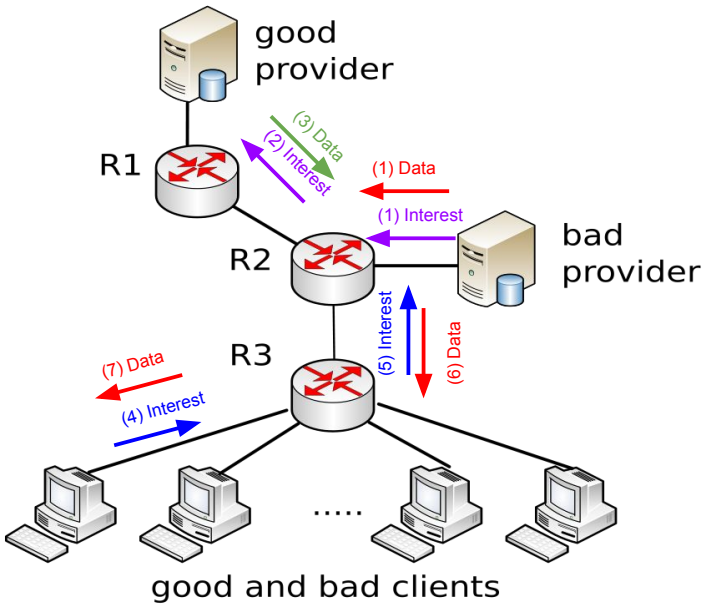


FIGURE 4.3 – Empoisonnement de cache en s'auto-répondant

vais clients envoient régulièrement des paquets *Interest* pour des noms de contenus ciblés mais excluent les copies actuelles des paquets *Data* dans le but de contourner le cache des routeurs. Cela force R3 à transmettre la requête à R2, qui va à son tour la transmettre au bon et mauvais fournisseur de contenus en accord avec la stratégie de routage. Par conséquent, un paquet *Data* est renvoyé par les deux fournisseurs de contenus. Mais, dû à un délai plus court, le mauvais paquet *Data* arrivera plus vite que le bon, il consommera donc l'entrée PIT associée et le paquet sera stocké en cache pour les routeurs R2 et R3. Pendant ce temps, le bon paquet *Data* sera jetté à cause de son arrivée tardive car il n'y aura plus d'entrée en PIT au niveau de R2 correspondant à ce paquet.

4.2.4 Scénario *Best route*

Best Route est la stratégie de routage par défaut utilisée par l'implémentation actuelle du routeur NDN [ASZ+14]. Un routeur qui utilise cette stratégie transmet les paquets *Interest* entrant à la *Face* qui a le coût le plus faible parmi celles correspondantes dans la FIB. Si il existe au moins deux *Faces* avec le même coût, le routeur choisira la première à s'être enregistrée. Après qu'un paquet *Interest* est routé, un paquet *Interest* similaire avec le même nom et les mêmes sélecteurs mais une nonce différente (valeur qui sert à éviter les boucles) sera ignoré si il est reçu durant l'intervalle de suppression de retransmission. Un paquet reçu après cet intervalle est considéré comme une retransmission, et sera routé sur la prochaine *Face* avec le coût le plus faible et ainsi offre une opportunité pour le mauvais fournisseur de contenus d'agir. Lorsque toutes les *Faces* valides pour un paquet *Interest* ont été utilisées, le paquet sera de nouveau routé sur la première. Grâce à la collaboration avec un utilisateur malveillant, l'attaquant peut générer plusieurs paquets *Interest* similaires, forçant le routeur R2 à utiliser d'autres routes menant vers le mauvais fournisseur de contenus, qui enverra en réponse de mauvais paquets *Data* dans les caches de R2 et R3.

Constante	Valeur
Nombre de contenus du bon fournisseur de contenus	10000 contenus
Fraîcheur des contenus du bon fournisseur de contenus	90 secondes
Latence réseau du bon fournisseur de contenus	100 microsecondes
Fraîcheur des contenus du mauvais fournisseur de contenus	120 secondes
Latence réseau du mauvais fournisseur de contenus	10 microsecondes
Taille du grand cache	1000 contenus
Taille du petit cache	500 contenus
Vitesse d'émission de l'utilisateur	10 *Interests*/seconde
Facteur de la loi Zipf	1.5
Nombre maximum de composants à exclure	700

TABLE 4.1 – Constantes expérimentales

4.3 Évaluation de l'attaque

Dans cette section, nous présentons le protocole utilisé pour nos expérimentations sur les bases de notre topologie, présentée plus haut, ainsi que les scénarios d'attaque que nous avons identifiés. Puis, nous évaluerons l'impact de ces attaques vu par le utilisateur, la cible principale, ainsi que par le fournisseur de contenus légitime et les routeurs comme ils seront aussi des victimes collatérales de l'attaque. Il est nécessaire de mentionner que sur nos figures, les couleurs rouge, verte et bleu sont respectivement attribuées aux scénarios *bestroute*, *multicast* et *unregistered remote provider*. Pour finir, en s'appuyant sur une analyse en composantes principales, nous révélons des modèles d'attaque synthétiques pour tous les scénarios.

4.3.1 Environnement expérimental

Durant nos expériences, nous déployons la topologie décrite sur la Figure 4.1 et utilisons la version 0.4.1 de NFD dans des conteneurs Docker, un conteneur par composant de la topologie. Nous avons défini une latence artificielle sur les deux fournisseurs de contenus pour émuler une distance réseau plus réaliste entre utilisateurs et serveurs. Durant nos expériences, tous les fournisseurs de contenus et utilisateurs sont connectés à distance à des nœuds NFD ainsi les caches de contenus ne seront présents que sur les routeurs NDN. Les paramètres constants de nos expériences sont listés dans la Table 4.1 et la plupart de ces valeurs sont motivées par [RR11]. Chaque expérience dure 600 seconds, avec les 300 premières secondes utilisées sans attaque dans le but de comparer les statistiques avant et après l'attaque. Pour finir, chaque expérience est lancée cinq fois et toutes les points des courbes représentés par la suite correspondent à la moyenne des 5 résultats délimités avec un intervalle de confiance de 95%.

Paramètres d'attaque

Nous considérons l'intensité de l'attaque comme le paramètre qui influe le plus sur le succès de celle-ci. Pour le scénario *unregistered remote provider*, ce paramètre représente le nombre de mauvais *Data* envoyés par seconde par le *unregistered producer*, celui-ci varie dans la plage $[10, 1000]$ suivant une échelle logarithmique et est fixée à 50 paquets *Data* par seconde comme valeur par défaut. Nous n'effectuons pas d'expérience pour la plage de 1 à 10 (inférieur à l'émission de paquets *Interest*) car la CPA est difficilement réalisable avec ce taux d'attaque. Pour les scénarios

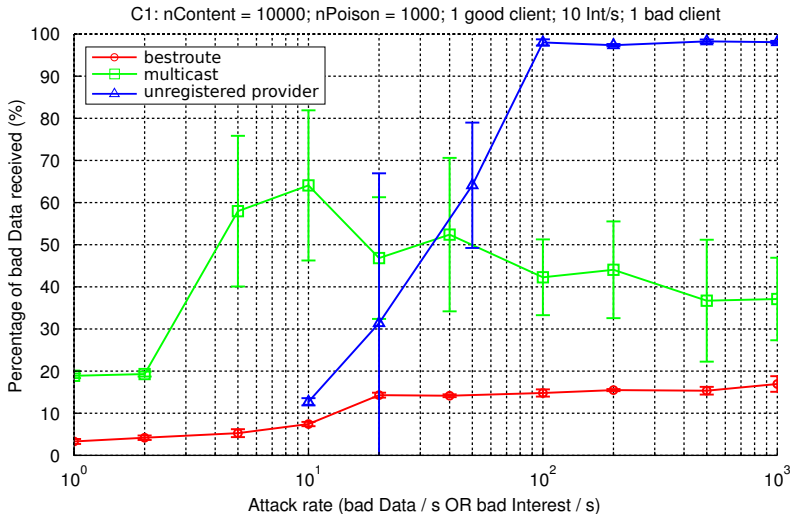


FIGURE 4.4 – Effet de l'intensité de l'attaque CPA sur l'utilisateur légitime

multicast et *best route*, il représente le nombre de mauvais paquets *Interest* injectés par seconde, celui-ci varie dans la plage $[1, 1000]$ suivant une échelle logarithmique et est fixée à 10 paquets *Interest* par seconde, c'est-à-dire le même taux qu'un bon utilisateur.

Comme second paramètre, nous considérons la fraction des contenus les plus populaires qui seront ciblés par la CPA. Cette valeur varie dans la plage $[0.01, 10]$ pourcent des contenus, suivant aussi une échelle logarithmique et est fixée à 1% comme valeur par défaut.

Comportement de utilisateur

Le comportement de utilisateur est implémenté comme décrit dans la section 2, et consiste à renvoyer un nouveau paquet *Interest* chaque fois qu'il reçoit un mauvais paquet *Data* en ajoutant le nom de ce paquet dans le champ *Exclude*. Cependant, l'utilisateur ne peut pas exclure des noms NDN indéfiniment à cause de la limitation de la taille des paquets défini par le protocole NDN. Par conséquent, l'utilisateur considérera un contenu inaccessible lorsque la taille *Exclude* atteint une valeur maximale et demandera ce contenu plus tard avec un nouveau champ *Exclude* vide. Nous avons décidé d'utiliser cette façon de procéder en lieu et place de la façon naïve (pas de mémoire des précédents essais) car l'utilisateur ne sait pas si les mauvais paquets précédemment *reçus* existent encore dans le réseau mais il ne veut pas les récupérer une seconde fois, c'est pourquoi il réutilisera le dernier champ *Exclude* qui a permis de récupérer un bon paquet *Data*.

4.3.2 Impact sur l'utilisateurs légitime

Les Figures 4.4 et 4.5 révèlent les effets de la CPA sur les utilisateurs légitimes. Ils représentent le pourcentage de mauvais paquets *Data* qu'un utilisateur légitime reçoit lors de ses demandes de contenu en fonction de l'intensité de l'attaque (Figure 4.4) et du nombre de contenus ciblés (Figure 4.5). La Figure 4.4 montre que dans le cas du scénario *best route*, l'utilisateur légitime souffre le moins de la CPA. L'efficacité augmente légèrement lorsque l'intensité de l'attaque est supérieure à celle du trafic légitime et semble inchangée pour des intensités plus importantes. Comme le fournisseur de contenus légitime est accessible sur la route au coût le plus faible, les requêtes sont toujours transmises en priorité sur celle-ci et la route vers le mauvais fournisseur de contenus n'est utilisée qu'en cas de retransmission. Cela laisse bien moins de chance au mauvais

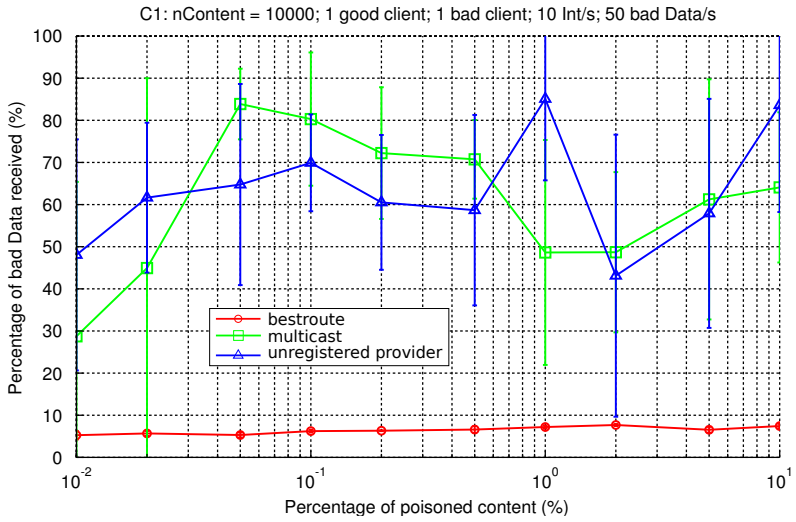


FIGURE 4.5 – Effet du nombre de contenus ciblés sur l'utilisateur légitime

fournisseur de contenus d'insérer des paquets *Data* dans le cache des routeurs. Ceci est rassurant car il s'agit du mode de routage par défaut. Les deux autres scénarios (*multicast* et *unregistered remote provider*) impactent l' utilisateur légitime bien plus efficacement. Plus particulièrement, dans le cas du scénario *unregistered remote provider*, avec une intensité d'attaque élevée, presque la totalité des paquets *Data* reçus sont mauvais. Sous une attaque soutenue, les paquets *Interest* entrant au niveau de R2 sont majoritairement répondus par de nouveaux mauvais paquets *Data* malgré l'effort d'exclusion. L'attaque *unregistered remote provider* qui, rappelons le est issue d'une faille de NFD, est donc clairement la plus dangereuse car facile à mettre en œuvre (pas besoin d'utilisateur malveillant) et d'une redoutable efficacité. D'un autre côté, les effets de l'attaque dans le cas du scénario *multicast* tendent à se réduire quand l'intensité de l'attaque augmente. Dans ce scénario, lorsque l'attaquant envoie trop de paquets *Interest*, plus de paquets *Data* légitimes seront stockés dans le cache de R1 et ce qui permettra à ces paquets d'avoir un meilleur délai que les paquets venant du mauvais fournisseur de contenus. Ainsi, lorsque qu'un paquet *Interest* est transmis par R2 sur les deux routes, les paquets *Data* légitimes ont plus de chance d'arriver en premier au niveau de R2.

La figure 4.5 montre que même lorsque l'attaquant change le nombre de contenus ciblés, le scénario *best route* garde sa protection relative contre la CPA. De plus, pour les scénarios *multicast* et *unregistered remote provider*, le nombre de contenus ciblés n'a pas l'air d'avoir un impact évident sur l'utilisateur légitime. Cela signifie que si un attaquant veut améliorer l'effet de son attaque sur les utilisateurs légitimes, il n'aurait pas besoin de s'efforcer pour augmenter le nombre de contenus à polluer car il suffirait de se concentrer sur les plus populaires.

4.3.3 Impact sur le fournisseur de contenus légitime

La figure 4.6 et 4.7 décrivent l'effet de bord de la CPA sur le fournisseur légitime. Plus spécifiquement, elles montrent le nombre moyen de paquets *Interest* que le fournisseur de contenus légitime doit traiter, en comparaison avec la phase sans attaque.

La Figure 4.6 montre que le scénario *unregistered remote provider* n'augmente pas la charge du fournisseur de contenus légitime quel que soit la force de l'attaque. Cela s'explique par la nature de cette attaque qui n'est pas basée sur l'émission de paquets *Interest* contrairement

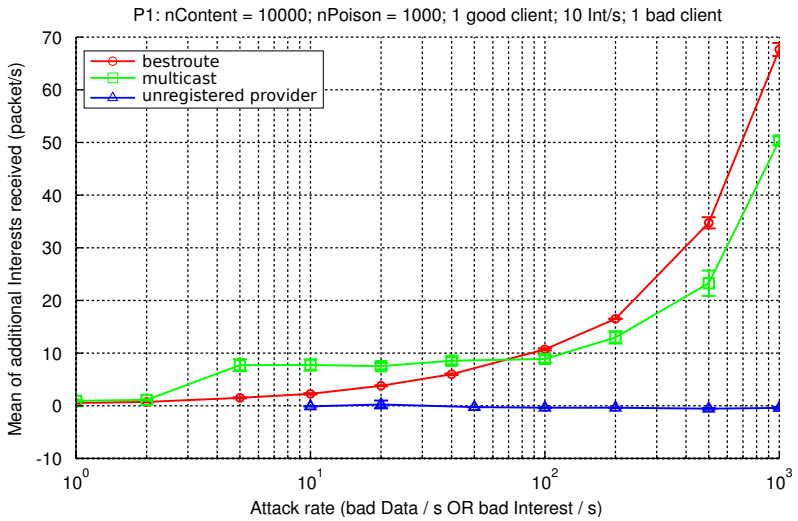


FIGURE 4.6 – Effet de l'intensité de l'attaque sur le fournisseur de contenus légitime

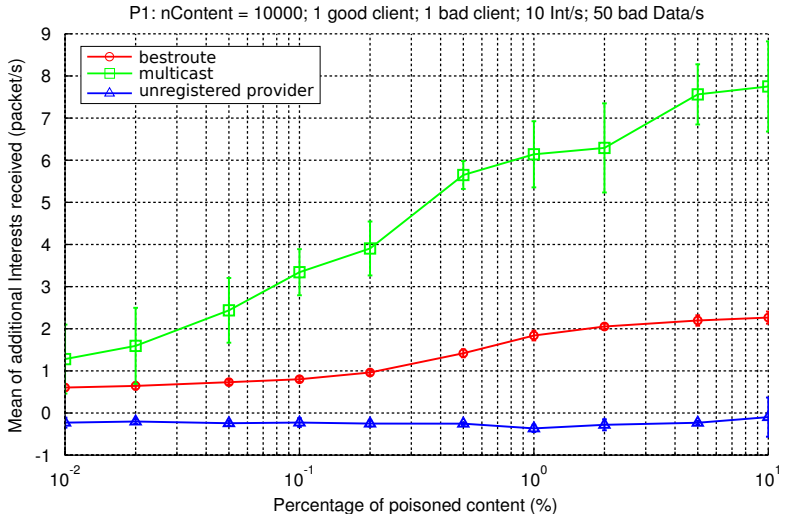


FIGURE 4.7 – Effet du nombre de contenus ciblés sur le fournisseur de contenus légitime

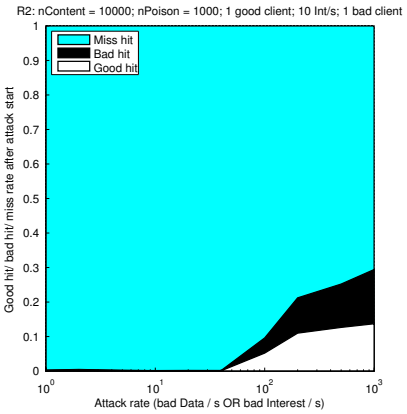
aux deux autres. Bien qu'un *unregistered remote provider* puisse consommer une entrée PIT au niveau de R2 avec ces mauvais paquets *Data*, le paquet *Interest* légitime qui l'a créé sera tout de même transmis au fournisseur de contenus légitime, même si le paquet *Data* résultant de cette requête sera ignoré (on envoie que des paquets *Data*, qui n'ont pas d'impact sur le routage des paquets *Interest*). Cela veut dire que le fournisseur de contenus légitime a moins d'indices pour détecter la CPA pour ce scénario que les autres éléments du réseau. A l'opposé, l'effet sur le fournisseur de contenus légitime est très corrélé avec l'intensité de l'attaque dans le cas des deux autres scénarios. Comme la plupart des paquets *Interest* envoyés par les mauvais utilisateurs sont reçus par le fournisseur de contenus légitime, celui-ci doit gérer plus de requêtes lorsque l'intensité augmente pour les deux scénarios *best route* et *multicast* qui montrent des tendances similaires.

La Figure 4.7 montre que le scénario *unregistered remote provider* n'a pas plus d'impact sur le fournisseur de contenus légitime quel que soit le nombre de contenus ciblés. Comme expliqué précédemment, cela est dû au fait que cette attaque ne se base pas sur l'émission de paquets *Interest* pour polluer des contenus mais envoie directement des paquets *Data* sans envoyer un paquet *Interest* pouvant correspondre. Pour les scénarios *best route* et *multicast*, la figure 4.7 montre une croissance presque linéaire du nombre de paquets *Interests* supplémentaires reçus par le fournisseur de contenus légitime selon le nombre de contenus ciblés. En effet, lorsqu'un mauvais utilisateur cible une gamme plus large de contenus, chaque paquet *Interest* émis a moins de chance de correspondre à une entrée PIT donnée car la gamme de noms est plus large, et par conséquent plus de paquets *Interest* sont transmis au fournisseur de contenus légitime.

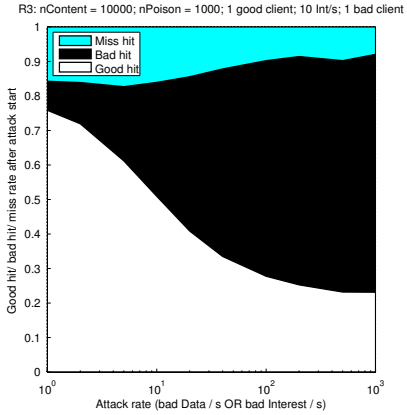
4.3.4 Impact sur le cache des routeurs

Cette section traite de l'impact de la CPA sur les routeurs les plus importants de la topologie : le routeur de cœur R2 et le routeur d'accès R3, comme illustrés sur la Figure 4.1. La mesure de l'effet de la CPA sur les caches du réseau est de première importance puisque les caches ont un rôle central pour éviter la congestion des réseaux NDN, mais aussi à cause de la grande quantité de ressources dédiées à cette fonction. Cela est encore plus critique si l'on considère que les attaquants peuvent exploiter les caches du réseau pour maintenir et amplifier la pollution à moindre coût. Les deux colonnes de la Figure 4.8 illustrent respectivement les mesures pour les routeurs R2 et R3. Les graphes montrent la proportion moyenne des *hits* de paquets *Data* légitimes, les *hits* des paquets *Data* du mauvais fournisseur de contenus et les *misses* (lorsqu'aucun paquet *Data* ne peut valider la requête de l'utilisateur) pour les caches des routeurs en fonction de l'intensité de l'attaque et du type de scénario. Les Figures 4.9 et 4.10 mesurent l'effet de l'intensité de l'attaque sur le taux d'insertion de mauvais paquets *Data* dans les caches des routeurs R2 et R3, respectivement.

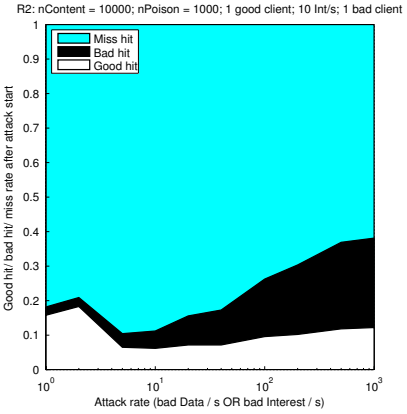
Nous considérons le routeur de cœur de réseau R2. Nous avons déjà identifié que même sans attaque, le cache de R2 n'est pas utile quand celui-ci est sollicité car l'on peut observer que même avec des attaques de faible intensité, le taux de *misses* est très important. La Figure 4.8a montre que l'attaque du scénario *best route* n'affecte pas le cache de R2 avec une faible intensité et a un impact limité avec une forte intensité avec seulement 10% de mauvais paquets *Data* provenant du cache. Cela peut s'expliquer par le fait que l'utilisateur récupère la majorité des mauvais *Data* de R3 et par conséquent, que la majorité des paquets *Interest* envoyés excluent déjà les noms de la plupart des paquets *Data* en cache ce qui empêche leur réutilisation. Dans le cas du scénario *multicast* (Figure 4.8c), l'effet est globalement plus important avec une proportion de mauvais paquets *Data* venant du cache plus importante et allant de 2% à 25% selon l'intensité de l'attaque. La stratégie *multicast* offre de meilleures opportunités au mauvais fournisseur de



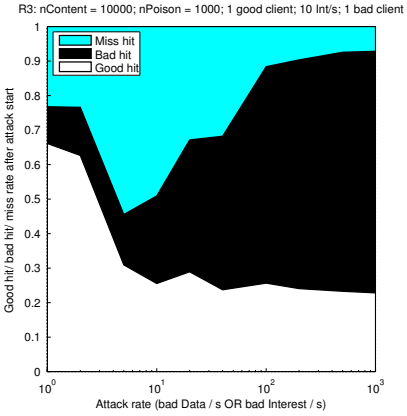
(a) R2 *hit* avec *best route*



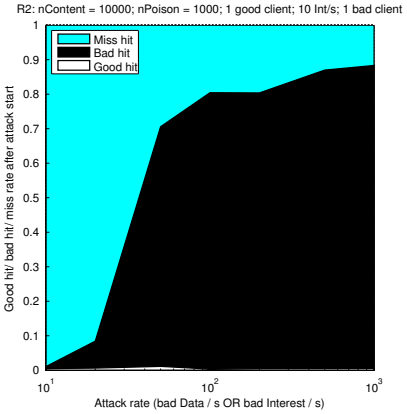
(b) R3 *hit* avec *best route*



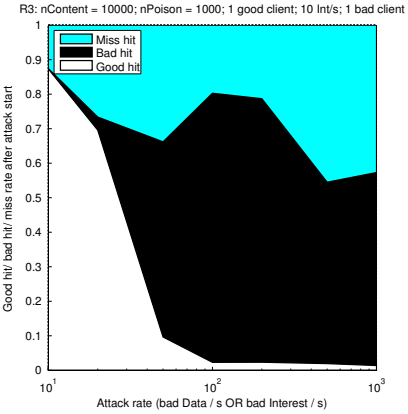
(c) R2 *hit* avec *multicast*



(d) R3 *hit* avec *multicast*



(e) R2 *hit* avec *unregistered remote provider*



(f) R3 *hit* avec *unregistered remote provider*

FIGURE 4.8 – Effet de l'intensité de l'attaque sur le routeur de cœur de réseau R2 (a)(c)(e) et le routeur d'accès R3 (b)(d)(f) pour chaque scénario d'attaque

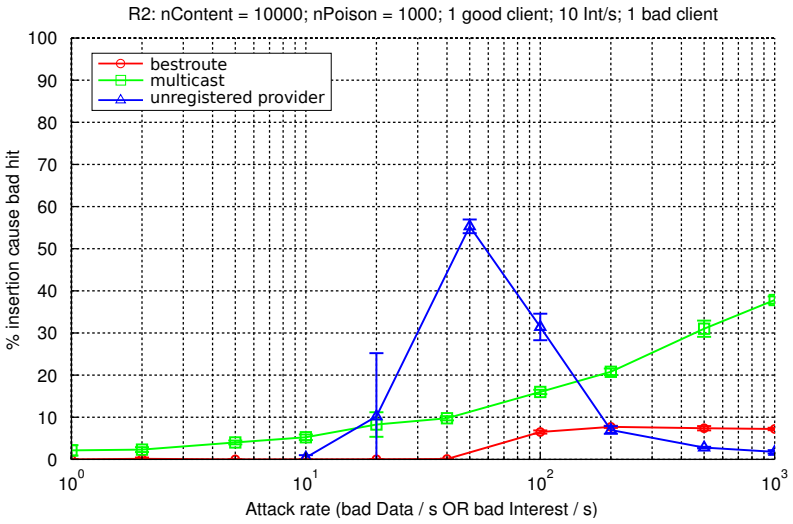


FIGURE 4.9 – Effet de l'intensité de l'attaque sur le routeur R2

contenus de répondre avec ses contenus ce qui peut expliquer ces résultats. Pour finir, l'attaque du scénario *unregistered remote provider* présente un comportement totalement différent sur R2 (Figure 4.8e). Le ratio de mauvais *hits* augmente rapidement avec l'intensité de l'attaque, atteignant 80% de mauvais *hits* avec une intensité de 100 mauvais paquets *Data* par seconde. Ensuite, augmenter l'intensité de celle-ci augmente peu l'impact de l'attaque avec seulement 5% de plus pour une attaque 10 fois plus agressive. Cette attaque est très efficace pour propager de mauvais contenus grâce au cache car le flux de mauvais contenus généré arrive facilement à s'insérer dans le cache en consommant les paquets *Interest* légitimes. Le fait que l'attaquant ait toujours une longueur d'avance sur l'utilisateur cherchant à exclure les noms des mauvais contenus explique également la proportion plus faible de *misses*.

En regardant l'aspect général des courbes de la seconde colonne de la figure 4.8, nous pouvons déjà voir que le cache du routeur d'accès R3 montre un comportement totalement différent du routeur de cœur R2 quand il est exposé à la même attaque. En premier lieu, R3 est plus à même de mettre en cache les paquets *Data* puisque l'on observe une grande proportion de *hits* sur les paquets *Data* légitimes lorsque l'attaque est de faible intensité. Puis, toutes les attaques montrent une tendance similaire avec la proportion de *hits* de paquets *Data* légitime qui décroît en faveur de mauvais paquets lorsque l'intensité de l'attaque augmente. Cet effet est néanmoins plus progressif dans le scénario *best route* (Figure 4.8b) que pour le scénario *multicast* (Figure 4.8d) qui ne fait qu'augmenter la proportion *misses* avec des attaques de faible intensité avant de d'augmenter la proportion de *hits* de mauvais paquets *Data* jusqu'à 500 paquets *Interest* par seconde. Dans les deux cas, au plus haut niveau d'attaque, la proportion de *hits* de mauvais paquets *Data* est très importante ($\simeq$70%) comparée à celle des paquets *Data* légitimes ($\simeq$20%) et de *misses* ($\simeq$10%), ce qui rend ces deux attaques très efficaces sur R3. La tendance décrite ci-dessus est atteinte plus vite dans le cas du scénario *unregistered remote provider*, mais au maximum d'intensité de l'attaque nous avons des proportions différentes des deux autres scénarios. Il n'y a presque plus de *hits* pour des paquets *Data* légitimes comme affiché sur la Figure 4.8f mais nous observons un certain équilibre entre les *hits* pour de mauvais paquets *Data* avec $\simeq$60% et les *misses* avec $\simeq$40%. Cependant, des intensités d'attaque plus faibles permettent d'avoir de meilleurs résultats pour cette attaque avec une proportion de *hits* sur de mauvais contenus allant jusqu'à 80% pour une intensité de 100 paquets *Data* par seconde.

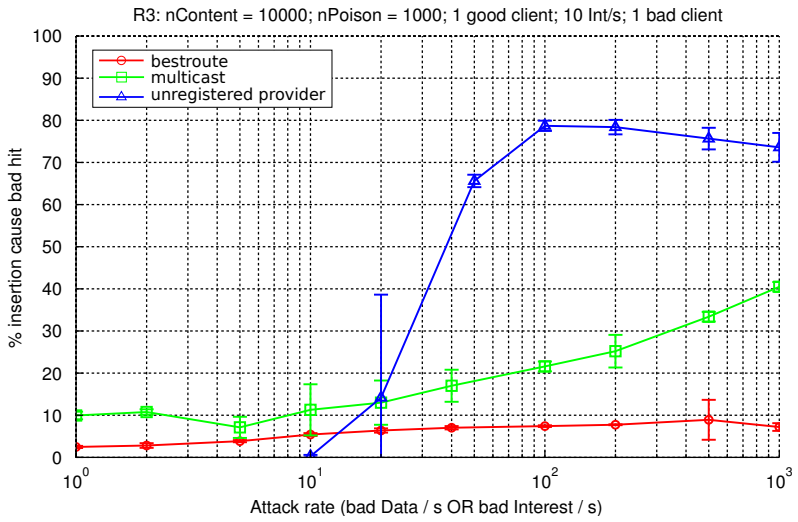


FIGURE 4.10 – Effet de l'intensité de l'attaque sur le routeur R3

Pour conclure sur l'intensité des attaques, nous pouvons dire que le scénario *unregistered remote provider* est plus efficace pour la pollution des caches des routeurs sur le chemin entre le utilisateur et le fournisseur de contenus légitime, mais son efficacité se réduit lorsque que l'on veut traverser plusieurs routeurs. Un scénario alternatif serait de placer le *unregistered remote provider* au niveau des routeurs d'accès pour reproduire le comportement du routeur R2 (Figure 4.8e) au plus proche des utilisateurs, ce qui rendra encore plus dur l'obtention de paquets *Data* légitimes. On peut aussi remarquer que la proportion de *misses* est plus importante sur R3 si l'on compare aux autres scénarios mais ils sont aussi plus susceptibles de générer un *hit* sur un mauvais contenu au niveau de R2. Concernant les scénarios *Best Route* et *Multicast*, ils ont moins d'impact sur le routeur de cœur R2 mais ont par contre un impact important sur le routeur d'accès R3.

La Figure 4.9 et 4.10 sont parfaitement en accord avec les résultats précédents. Par exemple, on peut remarquer un pic dans l'efficacité de l'attaque avec une intensité de 50 paquets *Data* par seconde pour le scénario *unregistered remote provider* sur le routeur R2, ce qui correspond au début du plat sur la Figure 4.8e. De même, l'efficacité maximale du scénario *unregistered remote provider* sur le routeur R3 est obtenu avec une intensité de 100 paquets *Data* par seconde et correspond à la plus grande proportion de *hits* de mauvais contenu comme observé sur la Figure 4.8f. Les scénarios *Best Route* et *multicast* ne montrent pas une efficacité maximum locale, dans le scénario *best route* celle-ci reste relativement constante avec l'intensité de l'attaque alors que pour le scénario *multicast*, l'efficacité augmente avec l'intensité de l'attaque. De manière générale, on peut conclure que le scénario *unregistered remote provider* est le plus efficace par paquet envoyé et cette faille devrait être rapidement corrigée par la communauté NDN.

4.3.5 Empreinte statistique des différents scénarios d'attaque

Pour finir, dans le cadre du suivi de l'analyse de données à variables multiples, Remi Cogranne a effectué une analyse en composantes principales (ACP) sur l'ensemble de nos données afin de révéler les corrélations de toutes nos mesures et paramètres. Les valeurs des deux premières composantes, qui représentent 80,5% de la variance totale des données, sont fournies sous forme de lignes dans la Table 4.2. Cette Table montre que la première composante, qui représente 56,5%

de la variance des mesures, est caractérisée par un impact élevé sur la proportion de *hits* sur de mauvais paquets *Data*, les ressources gaspillées pour les *hits* sur de mauvais contenus sur les deux routeurs, ainsi qu'un nombre important de mauvais paquets *Data* pour le bon utilisateur. À ce titre, cette première composante représente l'impact principal attendu de la CPA avec l'insertion de mauvais paquets *Data* dans le cache des routeurs. En parallèle, la deuxième composante principale, représentant 24% de la variance totale des données, montre un impact similaire sur le nombre de mauvais paquets *Data* pour l'utilisateur, mais un impact beaucoup plus élevé sur le taux de *miss* du routeur d'accès, dans une moindre mesure sur le routeur de cœur et sur le trafic supplémentaire vers le fournisseur. Ceci montre l'effet secondaire de la CPA qui empêche les routeurs de mettre en cache les paquets *Data* légitimes, créant ainsi un taux plus élevé de *miss* et de trafic vers le fournisseur de contenus légitime. La figure 4.11 présente la projection de chaque mesure sur ces deux premières composantes ainsi que la projection moyenne pour chaque scénario. La figure montre bien que les composantes distinguent le scénario *unregistered remote provider* des scénarios *multicast* et *best route* qui représentent le même mode de fonctionnement. Sur la Figure 4.11, le cercle en cyan met en évidence la projection des expériences avec le moins d'impact. De façon similaire, les flèches en pointillés indiquent la direction vers laquelle le résultat tend lorsque que l'on augmente l'intensité de l'attaque. La figure montre clairement que le scénario *unregistered remote provider* a une empreinte spécifique principalement capturée par la première composante principale. Comme attendu, pour ce scénario les mauvais paquets *Data* génèrent un taux important de *hits* de mauvais contenus. La figure montre aussi que les scénarios *best route* et *multicast* ont un impact similaire lorsque l'intensité de l'attaque augmente, mis en évidence par la seconde composante principale. En effet, ces scénarios créent une proportion importante de *miss* comme les utilisateurs légitimes essaient d'éviter les mauvais paquets *Data* venant des caches, et inversement, les mauvais utilisateurs essaient d'empêcher la mise en cache de paquets *Data* légitimes. Cela explique également le nombre plus important de requêtes reçues par le *provider* légitime. Enfin, on peut remarquer que les scénarios *best route* et *multicast* sont également, dans une moindre mesure, caractérisés par la première composante principale. Cependant, le scénario *best route* présente un impact beaucoup plus faible.

4.4 Solutions contre l'attaque par empoisonnement de contenus

4.4.1 Correction de la vulnérabilité de NFD (prévention)

Comme nous avons pu le voir durant les différentes expériences réalisées, l'attaque *unregistered remote provider* peut avoir un impact important sur l'infrastructure et les utilisateurs. Néanmoins, cette vulnérabilité, une fois bien identifiée, est relativement simple à corriger. Dans l'implémentation utilisée, il est possible de prévenir ce problème d'au moins deux manières différentes, en utilisant au choix : la liste des out-records contenue dans les entrées PIT ou bien la liste des *Faces* contenue dans les entrées FIB.

Dans le premier cas, il suffit de retirer de la liste des entrées PIT retournée après une correspondance (Listing 4.1, ligne 10) toutes les entrées dont le paquet *Interest* n'a pas été transmis via cette *Face*, grâce aux out-records stockés, car cela évoque alors une donnée non sollicitée et donc suspecte. Cette méthode réutilise les informations déjà récupérées durant la recherche dans la PIT et ne nécessite pas de nouvelle recherche. Cette solution ajoute ainsi la restriction telle qu'un paquet *Data* doit obligatoirement suivre le même chemin que le paquet *interest* correspondant même s'il existe plusieurs routes pour un nom donné. Une *Face* valide recevant un paquet *Data* et n'ayant pas transmis le paquet *Interest* au préalable verra ce paquet rejeté (ce cas n'est cependant pas sensé arriver en temps normal). Une implémentation de ce patch est disponible

TABLE 4.2 – Valeurs des deux premières composantes principales ainsi que le nom de leurs métriques

Provider's side	Core router's side				Access router's side.				Client's side	
# additional *Interests*	% good hit	% bad hit	% miss hit	Resources waste	% good hit	% bad hit	% miss hit	Resources waste	% bad *Data* received	# bad *Data*
-0.1618	-0.0778	0.3913	-0.3891	0.3036	-0.3554	0.2976	0.1227	0.4018	0.3243	0.2731
0.4252	0.3178	-0.144	0.1085	-0.1963	-0.24	-0.2521	0.5727	-0.0401	0.3626	0.2549

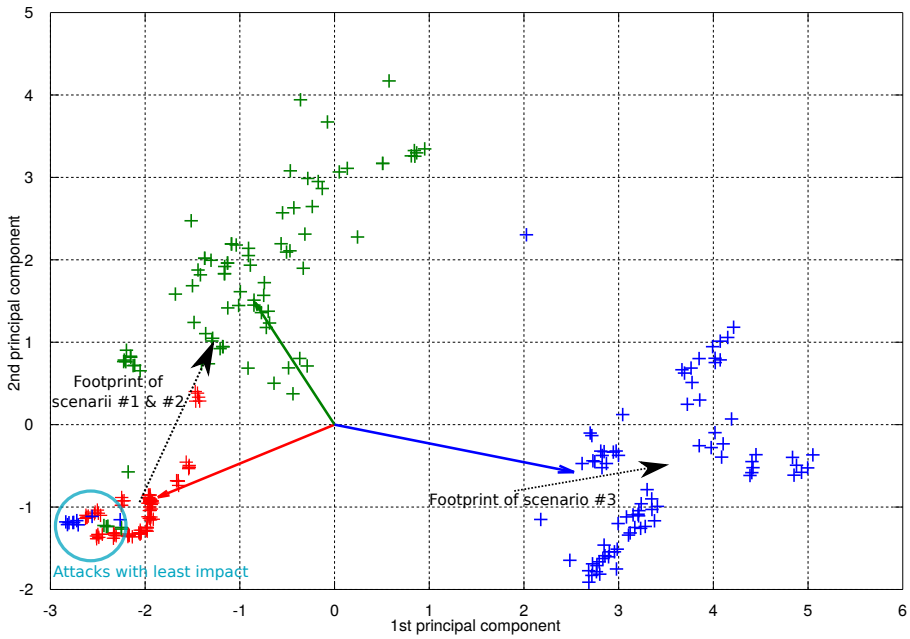


FIGURE 4.11 – Projection des mesures sur les deux premières composantes principales (les flèches continues représentent la projection moyenne pour chaque scénario et les flèches en pointillées montrent la direction lorsque l'intensité de l'attaque augmente)

dans le Listing 4.3.

Dans le second cas, il suffit de regarder dans la FIB avant de parcourir la PIT si la *Face* par laquelle le paquet *Data* est reçu a bien enregistré un préfixe pour ce paquet. Cette méthode a le mérite d'être plus souple mais va nécessiter deux parcours de table pour les paquets validant cette condition : la FIB puis la PIT. Bien qu'un peu plus coûteuse en temps processeur que la première solution si les deux parcours sont réalisés, cette méthode ne nécessite pas de stocker des informations supplémentaires (*stateless*) et celle-ci est applicable avant le parcours de la PIT, ce qui pourrait économiser des parcours inutiles si des paquets non sollicités sont reçus. Une implémentation de ce second patch est disponible dans le Listing 4.4.

Listing 4.3 – Exemple de patch basé sur les informations de la PIT

```
{...} //after PIT match check
auto it = pitMatches.begin();
while(it != pitMatches.end()){
  if((*it)->getOutRecord(inFace) == (*it)->out_end()){
    it = pitMatches.erase(it);
  } else{
    ++it;
  }
}
{...} //but prior Unsolicited Data pipeline
```

Listing 4.4 – Exemple de patch basé sur les informations de la FIB

```
{...} //some stuff and scope check
auto& fibEntry = m_fib.findLongestPrefixMatch(data.getName());
if (!fibEntry.hasNextHop(inFace)){
  return;
}
{...} //PIT match check, CS insert and so on...
```

Parmi ces deux solutions, la deuxième, basée sur la FIB, nous parait la plus judicieuse à implémenter. Tout d'abord, les in/out-records semblent être spécifiques à l'implémentation pour pouvoir gérer les NACK que le routeur reçoit. De plus, la première solution est basée sur des informations éphémères (*stateful*) stockées dans les entrées de la PIT qui nécessitent son parcours pour y accéder. La solution basée sur la FIB, quant à elle, est *stateless* et peut se placer avant le parcours de la PIT pour éviter des accès inutiles à la PIT en cas d'attaque, et ainsi contrebalancer la nécessité de parcourir les deux tables pour des paquets *Data* valides, la FIB étant plus petite que la PIT. Cette première solution pour contrer la CPA est d'autant plus importante qu'elle permet d'éviter le scénario *unregistered remote provider* que nous avions précédemment évalué comme étant le plus critique pour réaliser la CPA.

4.4.2 Détection et protection dynamique contre l'attaque par empoisonnement de contenus (réaction)

L'analyse en composantes principales nous a permis de distinguer les deux types d'attaque que nous avons mises en œuvre dans ce chapitre. Ainsi il serait possible de détecter ces deux types d'attaque selon les métriques identifiées, c'est ce que se sont proposés de faire d'autres membres du projet DOCTOR [MND+18]. Nos collègues de l'UTT ont ainsi proposé un plan de monitoring pour identifier des attaques comme celles présentées ici grâce à une sonde logicielle, développée par Montimage, un autre partenaire du projet DOCTOR, qui écoute les évènements

réseau ainsi que les logs de NFD. Des micro-détecteurs se chargeront d'analyser les différentes métriques remontées par la sonde et lever des alertes en cas de comportement anormal. Certaines attaques peuvent être identifiées via une métrique unique comme l'IFA (taille de la PIT) mais des attaques plus complexes comme la CPA ont besoin de plusieurs métriques pour être détectées. Ainsi, ils ont réalisé un réseau bayésien pour pouvoir établir des liens de causalité entre les alertes émises par les micro-détecteurs et ainsi correctement détecter ce type d'attaques. Leurs travaux montrent que la CPA est donc bien détectable, mais ils ne sont pas capables de connaître précisément le nom du contenu attaqué.

Dans le dernier chapitre (Section 6.6.4) nous compléterons la lutte contre la CPA grâce à un orchestrateur capable de contrer cette attaque en se basant sur des métriques remontées périodiquement par les caches du réseau et en déployant notamment, et à la volée, un module de vérification de signature où le problème a été localisé (et possiblement un *firewall* alimenté par ce module pour filtrer plus efficacement) grâce à une vue globale du réseau. Notre solution, dans sa partie détection par analyse des caches, est plus simple que celle présentée précédemment, mais a pour avantage de détecter le nom des contenus attaqués (ceux improprement signés). De plus, elle permet dans un second temps le déploiement de contre-mesures dynamiques pour remonter à la source de l'attaque et l'empêcher en filtrant les contenus malveillants.

4.5 Conclusion

En proposant une étude sur le comportement réel des acteurs d'un réseau NDN sous trois scénarios d'attaque que l'on a spécifiés et mis en œuvre, nous avons pu mettre en évidence des faiblesses critiques dans la conception du protocole NDN et l'implémentation de NFD qui peuvent être exploitées pour assurer le succès de la CPA. De plus, à travers de nombreux résultats d'expériences, nous avons prouvé dans quelle mesure la CPA est faisable dans la réalité. D'une part, l'attaque que nous avons découverte en nous appuyant sur le scénario *unregistered remote provider* a un faible effet sur le fournisseur, mais l'effet le plus élevé sur l'utilisateur légitime ainsi que sur les routeurs de cœur et d'accès, en particulier avec une attaque de forte intensité. Elle est probablement la plus simple a mettre en œuvre mais aussi à corriger avec un patch dédié dans une future révision de NFD. En attendant, elle constitue une réelle menace pour NDN. D'autre part, les scénarios *best route* et *multicast* sont plus difficiles à mettre en œuvre mais sont plus difficiles à contourner car ils se basent sur une utilisation normale du protocole NDN. Ces scénarios ont un impact plus faible côté utilisateur mais un impact plus important côté fournisseur de contenus. L'impact sur le fournisseur de contenus et sur le routeur d'accès augmente avec la force de l'attaque mais ne semble pas avoir d'effet côté utilisateur. Enfin, peu importe l'intensité de l'attaque, les utilisateurs sont bien protégés de la CPA dans le cas du scénario *best route*. Le fait que les attaques par empoisonnement de contenus soient contenues (patchs) ou puissent être contrées répond à la l'impératif de sécurisation du réseau.

A l'issue de cette seconde partie, nous avons vu que les problèmes de performances et de sécurité que nous avions identifiés comme non couverts par la littérature étaient avérés mais ont pu être en même temps grandement réduits à l'issue de notre étude. Néanmoins, garantir les performances et la sécurité d'une nouvelle architecture réseau n'est pas suffisant pour promouvoir celle-ci au stade du déploiement. Il faut en effet que les applications existantes puissent en tirer profit afin de générer les données à transporter. Dans la troisième partie cette thèse nous allons donc nous intéresser à l'interopérabilité entre le web et NDN, ce qui permettra de définir notre vision du déploiement de NDN par îlots, puis nous terminerons par notre solution basée sur NFV

permettant de déployer et de gérer efficacement un tel réseau.

Troisième partie

Interopérabilité et déploiement

Chapitre 5

Transporter HTTP sur NDN : conception d'un protocole d'adaptation et évaluation de sa mise en oeuvre par des *gateways*

Sommaire

A noter : ce travail a été réalisé en collaboration avec Orange Labs et l'Université Technologique de Troyes (UTT), notamment avec les personnes suivantes : Bertrand Mathieu, Antoine Saverimoutou et Guillaume Doyen.

5.1 Introduction

Migrer d'un protocole réseau standard et massivement déployé à un nouveau est connu pour être une opération très lente qui repose sur le déploiement de solutions techniques peu commodes comme le déploiement d'équipement implémentant les deux piles protocolaires distinctes

ou l'encapsulation complète du protocole dans un tunnel réseau pour assurer la connectivité durant la migration. D'un autre côté, les spécifications de NDN et son logiciel NFD [ASZ+15] sont maintenant assez matures pour pouvoir être déployés et ainsi les travaux de recherche pourraient profiter d'un déploiement à large échelle de ce protocole transportant du trafic d'utilisateurs réels, pour par exemple, évaluer la proportion de trafic qu'il est possible d'économiser en utilisant un réseau NDN à la place de l'actuel utilisant IP. Quelques applications natives ont déjà été mises à disposition pour le protocole NDN comme la possibilité de diffuser des vidéos (à la manière du multicast) ou encore le transfert de fichiers, mais elles ont besoin d'un client et d'un serveur compatibles avec le protocole NDN ce qui réduit drastiquement les utilisateurs potentiels et limite la taille de la plateforme de test. Comme les fournisseurs d'accès sont les principaux intéressés car ils pourront profiter des avantages liés au déploiement du protocole NDN, pour notamment réduire leur coût d'infrastructure et d'investissement, nous pensons que NDN doit être transparent pour les utilisateurs finaux (consommateurs et producteurs de contenus), au moins durant les premières phases de déploiement et des efforts doivent être engagés pour proposer à la communauté réseau des composants pour pouvoir transporter le trafic sur NDN de manière efficace et intelligente.

C'est pour cette raison que nous avons développé un mappage HTTP/NDN ainsi que son architecture, dont l'implémentation prend la forme de *gateways* HTTP/NDN qui peuvent être utilisées pour transporter de manière transparente le trafic HTTP dans un îlot NDN. Notre solution a été développée avec plusieurs objectifs en tête, organisés selon quatre points clés :

1. Transparence : être transparente pour les utilisateurs web non NDN (clients et serveurs, pas d'installation requise) et qui préserve les fonctionnalités d'HTTP (ou du moins les plus importantes) ;
2. Réciprocité : autoriser les clients web non NDN à communiquer avec des serveurs natifs NDN, et inversement, pour faciliter la transition d'IP vers NDN ;
3. Mise en cache : transporter le protocole HTTP via NDN de façon à pouvoir profiter des fonctionnalités de NDN et en particulier celle de mise en cache ;
4. Performance et fiabilité : être efficace (pas d'erreurs ni de service dégradé) et atteindre de bonnes performances (débit et délai).

Ces objectifs sont ambitieux mais néanmoins nécessaires pour que les *gateways* soient efficaces et utilisables comme composants principaux dans le déploiement progressif du protocole NDN et permettent la diffusion de contenus venant d'Internet à travers ce type de réseau. Dans ce chapitre, nous présenterons nos choix de conception de nos *gateways* et décrirons comment nous transportons HTTP sur NDN de manière efficace. Nous proposerons aussi une évaluation de notre solution à travers un jeu d'expériences complet.

Le reste du chapitre est organisé comme suit : La section 5.2 décrit notre mappage et notre architecture. La section 5.3 enquête sur la question de la mutualisation des requêtes pour optimiser le transport du protocole HTTP sur NDN. L'implémentation des *gateways* est présentée et évaluée dans la section 5.4. Puis, la section 5.5 conclura ce chapitre.

5.2 Notre architecture pour le transport de HTTP sur NDN

Nous considérons le web comme notre principale champ d'application à cause de sa grande popularité et son actuel impact sur le trafic Internet. En effet, les bénéfices de transporter le trafic web grâce aux protocoles de la famille des ICN ont déjà été démontrés depuis les origines de cette famille de protocoles, et se basent sur diverses propriétés comme les gains de l'encapsulation de

protocoles, les communications multicast et le cache de contenus dans les nœuds du réseau pour faciliter leur dissémination. Cependant, les clients et serveurs web n'implémentent pas encore ce type de protocole. Ainsi, nous avons conçu et développé une architecture composée de *gateways* ainsi qu'un protocole de mappage pour effectuer la conversion HTTP/IP vers HTTP/NDN.

5.2.1 Aperçu de l'architecture fonctionnelle

Un îlot NDN utilisant notre architecture est illustrée sur la Figure 5.1. Celle-ci a besoin de deux types de *gateways* :

1. Une *gateway* d'entrée (iGW), dont le but est de convertir les requêtes HTTP des utilisateurs en messages NDN mais aussi de convertir les messages NDN reçus en réponses HTTP qu'elle renverra aux utilisateurs en ayant fait la demande ;

2. Une *gateway* de sortie (eGW), le miroir de la première, qui va traduire les messages NDN en requêtes HTTP et les envoyer aux bons serveurs IP, puis convertir les réponses HTTP en messages NDN.

L'ensemble peut être considéré comme un proxy HTTP pour les clients et serveurs à l'extérieur de l'îlot, les *gateways* représentant les entrées et sorties de l'îlot NDN qui peut aussi stocker les réponses HTTP qui le traverse. De plus, des clients et serveurs web natifs NDN peuvent aussi être présents à l'intérieur de cet îlot, et peuvent communiquer avec des clients ou serveurs du monde IP en utilisant le même protocole de mappage que celui utilisé par les *gateways*.

La *gateway* d'entrée est composée de trois modules comme illustrés sur la Figure 5.2 (à gauche). Un module « HTTP server » gère les connexions TCP/IP et analyse les entêtes HTTP des requêtes envoyées par les utilisateurs. Puis un second module, nommé « interpreter », va traiter les requêtes reçues du premier module, en changeant au besoin certains éléments de l'entête dans un besoin d'unification des requêtes. Il va aussi générer un nom NDN pour chaque requête et enverra le tout à un troisième module (uniquement si le nom NDN n'est pas déjà en cours de traitement). Ce dernier module, nommé « NDN resolver », va gérer la communication sur le réseau NDN. Lorsque qu'un message est récupéré par le troisième module, il l'envoie au deuxième pour qu'il puisse vérifier qu'il s'agit bien d'une réponse HTTP et la traiter. Une fois analysée, elle sera transmise au premier module pour l'envoi à ou aux utilisateurs concernés.

La *gateway* de sortie est construite de la même façon que la *gateway* d'entrée. Le module « NDN resolver », récupère des messages NDN contenant les requêtes des utilisateurs et les envoie au module « interpreter ». Ce module, comme pour la *gateway* d'entrée, vérifie qu'il s'agit bien d'une requête HTTP et la traite avant de l'envoyer au dernier module, nommé « HTTP client » qui va se charger de communiquer avec le bon serveur et récupérer la réponse HTTP. Le chemin de retour est par contre un peu différent. Le module « interpreter » va regarder l'entête de la réponse pour définir quelques informations qui seront contenues dans les paquets NDN générés, comme la « fraîcheur » basée sur le champ HTTP « cache-control ». Pour finir, ces informations sont données au premier module pour qu'il puisse distribuer ce contenu. Celui-ci est aussi stocké dans un cache interne à la *gateway* avec une granularité différente du cache du réseau NDN (par contenu HTTP et non par paquet) pour pouvoir le redistribuer à d'autres pendant une courte période.

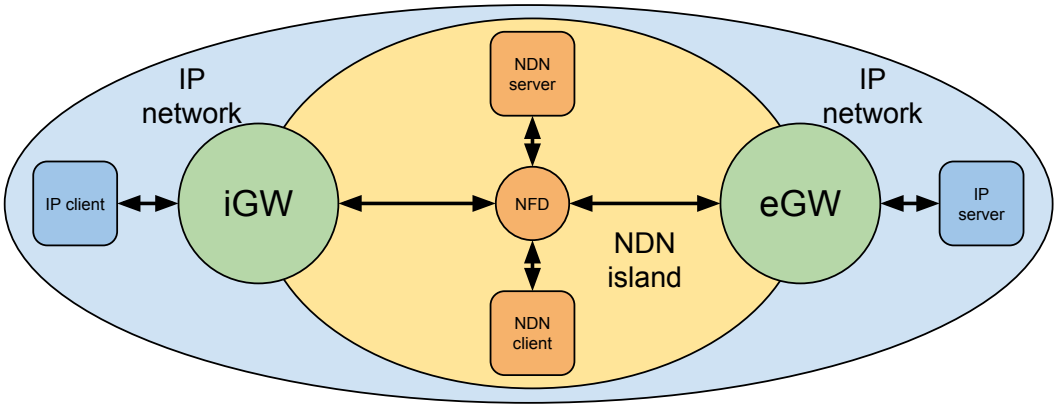


FIGURE 5.1 – Îlot NDN suivant notre architecture HTTP/NDN

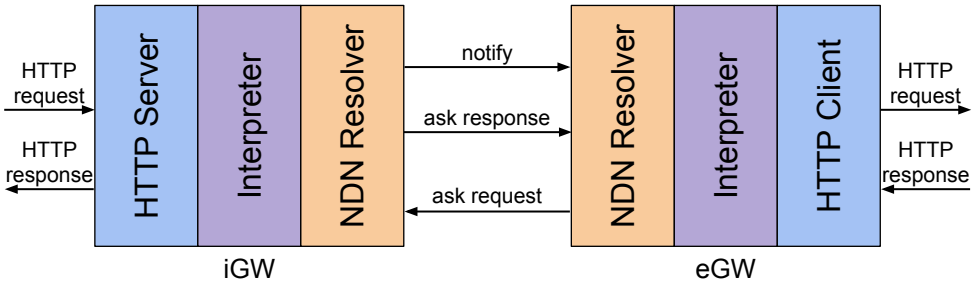


FIGURE 5.2 – Structure interne des *gateways*

5.2.2 Stratégie de nommage pour le transport de HTTP sur NDN

Pour communiquer entre elles, les *gateways* suivent un schéma de nommage basé sur une proposition de nommage pour convertir des URL en noms ICN [15] qui consiste à découper le domaine par sous-éléments et en les ajoutant dans l'ordre inverse pour obtenir un meilleur routage NDN en comparaison d'un nom constitué d'un domaine laissé sous sa forme « monolithique » comme utilisé dans [WBW+12] et [STC+13]. De plus, en ajoutant un préfixe « /http », une *gateway* de sortie peut enregistrer cette unique composant pour devenir le producteur de contenus par défaut pour tous les messages utilisant le protocole HTTP. Des noms plus précis peuvent aussi être utilisés comme par exemple « /http/fr » pour définir une *gateway* de sortie par défaut pour tous les domaines enregistrés en .fr. De plus, commencer le nommage par le type de protocole peut plus tard permettre de faire un routage par protocole et ainsi mieux gérer et optimiser le routage pour chacun d'entre eux en leur appliquant par exemple des stratégies de routage différentes, ce qui est prévu en 5G.

Comme les paquets *Interest* d'NDN ne sont pas capables de transporter des données alors qu'une requête HTTP le peut, la *gateway* d'entrée (ou un client respectant le mappage) doit échanger différents messages en trois étapes, définies dans la Table 5.1, pour récupérer le contenu web. La première étape consiste à envoyer un paquet *Interest* dont le nom contient, comme illustré dans la Table 5.1.a. :

1. Le domaine du serveur (fonctionne aussi avec les adresses IP) découpé et dont les éléments sont mis bout à bout dans l'ordre inverse (par exemple : www.google.com devient /com/google/www) ;

15. http ://www.icn-names.net/

Table 5.1 – Schéma de nommage utilisé pour le mappage

a.	/http/reverse_splitted_domain_name/path/sender_route/sha1
b.	/sender_route/sha1(/segment)
c.	/http/reverse_splitted_domain_name/path/sha1(/version/segment)

Table 5.2 – Exemple d'une requête HTTP en utilisant le mappage

a.	/http/com/firefox/detectportal/success.txt/%07%0B%08%04http%08%03iGW/1E69...
b.	/http/iGW/1E69...(/segment)
c.	/http/com/firefox/detectportal/success.txt/1E69.../%FCY%19%D1%98(/segment)

2. Le chemin du contenu ;

3. Le nom réseau de l'émetteur de ce paquet en tant que simple composant binaire comme défini par le format TLV (utilisé pour pouvoir demander la requête) ;

4. Un hash de l'entête de la requête HTTP (un SHA1 de l'entête et jusqu'à 1024 bytes du corps de la requête HTTP) pour parfaitement identifier les différentes requêtes pour une même URL et leurs paquets *Data*.

Ce paquet *Interest* est envoyé sur le réseau NDN pour demander à une entité si elle est capable de résoudre cette requête. Ainsi, la *gateway* de sortie (ou n'importe quel serveur natif qui respecte le mappage) sait à la réception de ce paquet que quelqu'un a une requête à satisfaire, avec le nom NDN pour l'atteindre (avant dernier élément du nom), et peut ainsi lui demander les « détails » de cette requête. Il faut savoir que le hash contenu dans le nom ne sert qu'à garantir que tous les paramètres présents dans les champs de l'entête de la requête HTTP sont pris en compte (user-agent, etc.). Choisir minutieusement un sous-ensemble de champs à considérer pour le calcul du hash peut améliorer le cache des contenus web dans le réseau NDN tout en donnant un résultat cohérent aux utilisateurs. Ceci sera approfondi dans la prochaine section.

Durant la seconde étape, la *gateway* de sortie extrait du premier paquet *Interest* envoyé par la *gateway* d'entrée les deux derniers composants, à savoir le nom au format binaire et le hash (utilisé comme un ID dans ce sens), pour pouvoir générer un nom pour demander à son tour la requête HTTP complète (Table 5.1.b) au client. Une fois cette étape terminée, la *gateway* de sortie peut demander la réponse au serveur IP.

Une fois la réponse reçue, la *gateway* de sortie va la découper en plusieurs *chunks*, représentés par les paquets NDN *Data*, et va utiliser un nom qui ressemble au tout premier paquet *Interest* mais en enlevant le préfixe de l'émetteur « sender route » (Table 5.1.c). Si l'on suit les principes de NDN, c'est au client de demander la réponse à l'aide de paquets *Interest*. Un exemple de notre mappage se trouve dans la Table 5.2 pour l'URL "http ://detectportal.firefox.com/success.txt"

Les *gateways* n'attendent pas la complétion des messages HTTP dans le monde IP pour commencer à effectuer le transfert dans l'îlot NDN. Après réception d'un entête valide, le message HTTP est streamé afin de ne pas induire une latence globale proportionnelle à la taille du message (1 RTT par requête HTTP au lieu de 1 RTT par *chunk*). Ainsi, nous avons un délai additionnel constant dû à la traversé de l'îlot plutôt que proportionnel à la taille du contenu.

5.2.3 Comparaison avec d'autres schémas de communication

D'autres protocoles de communication ont été proposés pour NDN pour qu'un fournisseur de contenu puisse obtenir des données de la part du client [MSO14]. Selon les auteurs, tous les shémas considérés ont des désavantages « significatifs ». Nous pouvons comparer notre shéma, dont le diagramme de séquence est donné sur la Figure 5.3, avec le shéma « Data Locator » et dans une moindre mesure le shéma « Application Data » présentés dans [MSO14]. La principale différence par rapport au shéma « Application Data » est que notre « jeton » est généré par l'utilisateur et est représenté par un hash de la requête. De plus, ce jeton est ajouté à la fin du nom NDN des paquets *Interest* au lieu d'un champ spécifique qui demanderait de modifier le format du paquet. Concernant le shéma « Data Locator », nous avons comme différences (1) que le nom NDN de l'émetteur est aussi placé dans le nom du premier paquet *Interest* au lieu d'un autre champ spécifique comme dans le précédent shéma ; (2) que notre protocole respecte la séquence des messages HTTP (voir Figure 5.3) alors que le leur consiste en une communication bidirectionnelle sans séparation claire entre requête et réponse.

Pour finir, notre modèle en trois étapes suit mieux les spécifications du protocole NDN et peut être réalisé en deux RTT car certaines étapes sont parallélisables (par exemple les messages 2 et 3 sur la Figure 5.3). Par contre, notre shéma est vulnérable aux mêmes scénarios de perturbation listés pour le shéma « Data Locator » : un attaquant peut envoyer des paquets *Interest* (message 1) avec des valeurs aléatoires de hash pour provoquer une émission de paquets *Interest* de la part de la *gateway* de sortie pour un nom NDN donné. Mais nous pensons que ce genre de cas peut facilement être détecté au niveau de la *gateway* de sortie si le client ne répond jamais aux paquets *Interest*.

Concernant la mobilité des utilisateurs, notre shéma n'est pas étudié pour répondre à cette contrainte. Ainsi si la *gateway* de sortie n'arrive pas à récupérer la requête avant l'indisponibilité de l'utilisateur, elle sera annulée et il devra réitérer les 2 premières étapes. Ce n'est par contre pas le cas de la troisième étape tant que le contenu est « disponible ». Nous avons aussi réfléchi sur une façon de réaliser des requêtes de manière implicite, en envoyant seulement la troisième étape de notre shéma avec un hash défini à « 0 », mais nous ne l'avons pas implémentée car cela augmente inutilement la complexité du code au vu de notre cas d'utilisation, mais cela pourrait résoudre le problème de mobilité pour des requêtes qui ne sont pas spécifiques à un utilisateur (utilisation de cookies, etc.).

5.2.4 Limites inhérentes

Comme le contexte utilisé pour le cache n'est pas le même entre HTTP et NDN, i.e l'un travaillant par contenu et l'autre par paquet, stocker des contenus HTTP dans un cache NDN souffre de limites inhérentes qui peuvent poser quelques soucis. Si l'on considère le cas où un *Content Store* ne possède pas tous les paquets *Data* d'un contenu spécifique et que celui-ci n'est plus disponible dans le « tampon » de la *gateway*, on peut identifier au moins deux causes d'échec possible du transfert HTTP :

1. Si le contenu change entre-temps : Il y aura un problème de version au premier essai car le cache ne contiendra pas tous les segments de la réponse et l'on ne peut pas spécifier la version que l'on veut sur IP.
2. Si le contenu ne change pas mais est de grande taille : Les *chunks* déjà en cache seront récupérés rapidement par la *Gateway* d'entrée jusqu'à un moment où les paquets *Interest* attendront un paquets *Data* de la part de la *Gateway* de sortie, ce qui peut provoquer

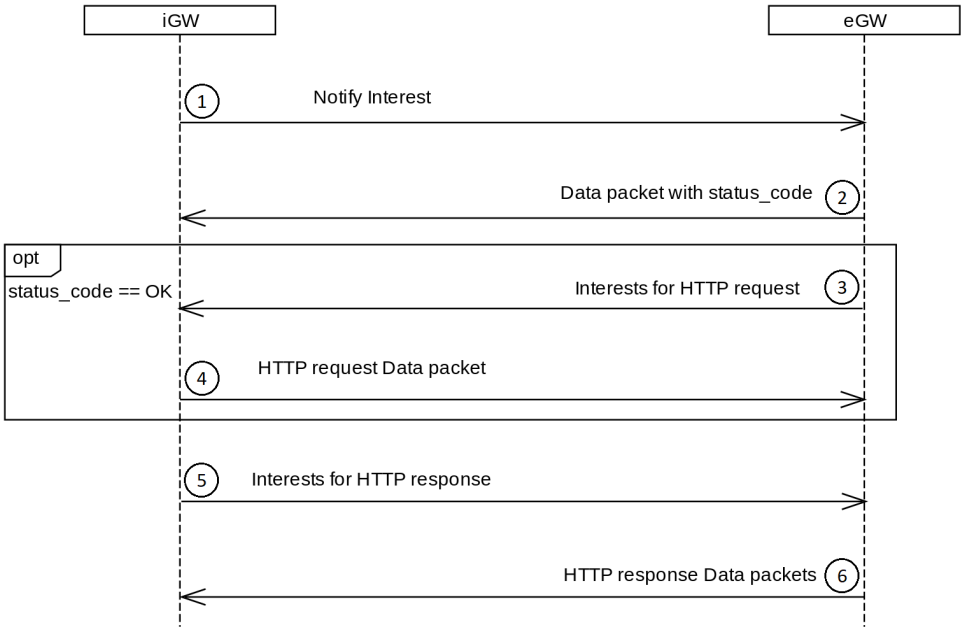


FIGURE 5.3 – Diagramme de séquence de la communication entre les *gateways*

un *timeout* car elle n'est pas capable de télécharger aussi vite le contenu demandé que le cache du réseau peut le servir.

Pour éviter les scénarios identifiés il faudrait, par exemple, que le *Content Store* des nœuds NDN utilise un unique *timer* pour tous les segments NDN *Data* d'un même contenu au lieu d'en avoir un pour chacun ce qui permettrait d'avoir la même granularité de traitement que HTTP.

5.3 Optimisation du transport de HTTP sur NDN

5.3.1 Mise en évidence du problème

Des expériences préliminaires (Table 5.3, première ligne « No unification ») montrent que, lorsque que l'on visite les pages d'accueil des 40 sites HTTP les plus populaires pour la première fois (le cache du navigateur est désactivé et le cache du réseau NDN est vide), le cache du réseau n'est presque jamais utilisé, comme attendu et cette valeur constitue notre mesure de référence. Cela peut s'expliquer car les contenus demandés sont tous différents ou appelés d'une manière différente. Par exemple, si l'on considère un script comme « jQuery.min.js » disponible dans un CDN, le champ *referer* de l'entête HTTP sera différent pour chaque page web qui le demande. Ensuite, si l'on redemande les même contenus une seconde fois (en rechargeant la page), les résultats montrent qu'un même utilisateur peut profiter du cache du réseau NDN pour une grande proportion de contenus (75,8%), ce qui est satisfaisant. On peut aussi voir qu'un utilisateur différent qui utilise la même configuration pour son navigateur peut aussi profiter des contenus en cache qui ont été demandés par le premier utilisateur mais dans une moindre mesure (38.3%). Cependant, un utilisateur qui utiliserait un navigateur différent (Firefox pour les deux premiers et Chrome pour le troisième) ne peut pas profiter des contenus en cache contrairement aux premiers. Cela est majoritairement dû au fait de la relative unicité des requêtes HTTP envoyées par un navigateur (les services web actuels veulent pouvoir identifier les utilisateurs). Ainsi, le cache NDN ne peut être utilisé que si un même contenu est demandé de la même façon

TABLE 5.3 – Pourcentage de paquet réutilisés du cache NDN lors de plusieurs visites avec différents utilisateurs (le pourcentage fait référence au nombre de paquet et non de contenus)

	Reference	Reload	New user Same browser	New user Different browser
No unification	0.2%	75.8%	38.3%	0.0%
Unification excluding cookie	0.2%	76.8%	37.8%	27.4%
Unification including cookie	0.2%	74.6%	78.3%	61.3%

(entête de la requête HTTP). Ce dernier résultat n'est pas satisfaisant et peut mettre en danger la capacité du trafic HTTP à pouvoir profiter des fonctionnalités de l'îlot NDN. Comme nous utilisons un hash SHA1 pour différentier les requêtes des utilisateurs, une possible solution serait d'ignorer certains champs de l'entête HTTP voir même d'en remplacer certains avec des valeurs par défaut, dans le but de maximiser l'utilisation du cache pour différentes requêtes sur un même contenu.

5.3.2 Constitution d'un jeu de données de requêtes HTTP

Comme les contenus statiques sont ceux qui vont le plus profiter de la capacité de cache du réseau NDN, nous avons décidé de nous focaliser sur ce type de contenu. Pour collecter quelques requêtes et réponses HTTP sur lesquelles nous pourrons évaluer différente stratégies, nous avons utilisé le navigateur Firefox pour charger la page d'accueil des 1000 sites HTTP les plus populaires et nous avons capturé le trafic qu'il a généré. Puis nous avons seulement gardé les requêtes HTTP pour des contenus qui peuvent être considérés comme statiques en se basant sur leur extension (nous utilisons une liste d'extensions donnée par Cloudflare [16]). Nous avons aussi fait de même avec un autre navigateur (Chrome) pour pouvoir les comparer. Les Figures 5.4 et 5.5 donnent le nombre d'occurrences pour chaque champ contenu dans l'entête de la requête HTTP. Quelques champs importants sont présents dans presque toutes les requêtes sélectionnées : *accept-language*, *accept*, *user-agent*, *accept-encoding*, *host*, *connection*, et d'autres comme *referer* ou *cookie* sont régulièrement utilisés, alors que certains champs semblent beaucoup rares et plus spécifiques.

5.3.3 Unification des requêtes HTTP dans le réseau NDN

Pour mener à bien l'unification des requêtes entre navigateurs, nous avons besoin de comparer les similarités entre les deux data sets (Firefox et Chrome) pour les champs de l'entête HTTP que nous avons choisis de considérer. Nous définissons le pourcentage de similarité entre deux ensembles de requêtes HTTP comme la taille de l'ensemble résultant de l'intersection des deux ensembles divisée par la taille du plus petit des deux ensembles. La Figure 5.6 donne la similarité des deux data sets après l'exclusion des différentes combinaisons possibles des huit champs les plus importants que nous avons mis en évidence à l'exception du champ *host* qui a toujours la même valeur pour un domaine donné. Sans aucune modification, tous les entêtes envoyés par Chrome sont différents de ceux envoyés par Firefox et cela sera le cas jusqu'à que l'on ignore/retire/remplace au moins trois champs lors du calcul du hash. Les data sets commencent à montrer des requêtes similaires lorsque l'on édite les trois champs *user-agent*, *accept-language* et *accept-*

16. https ://support.cloudflare.com/hc/en-us/articles/200172516-Which-file-extensions-does-Cloudflare-cache-for-static-content-

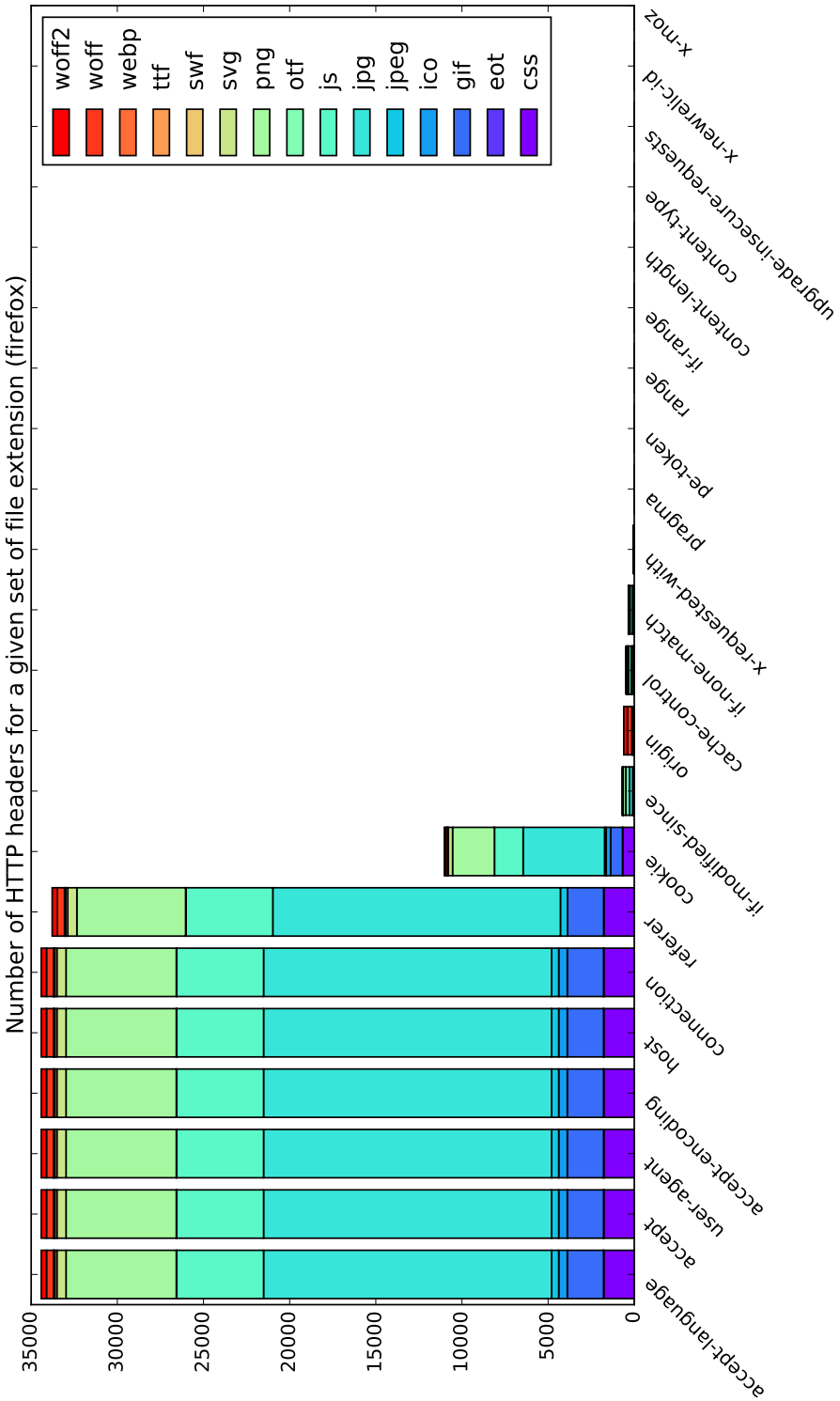


FIGURE 5.4 – Nombre d'occurrences de chaque champ de l'entête HTTP pour chaque extension de fichier avec Firefox

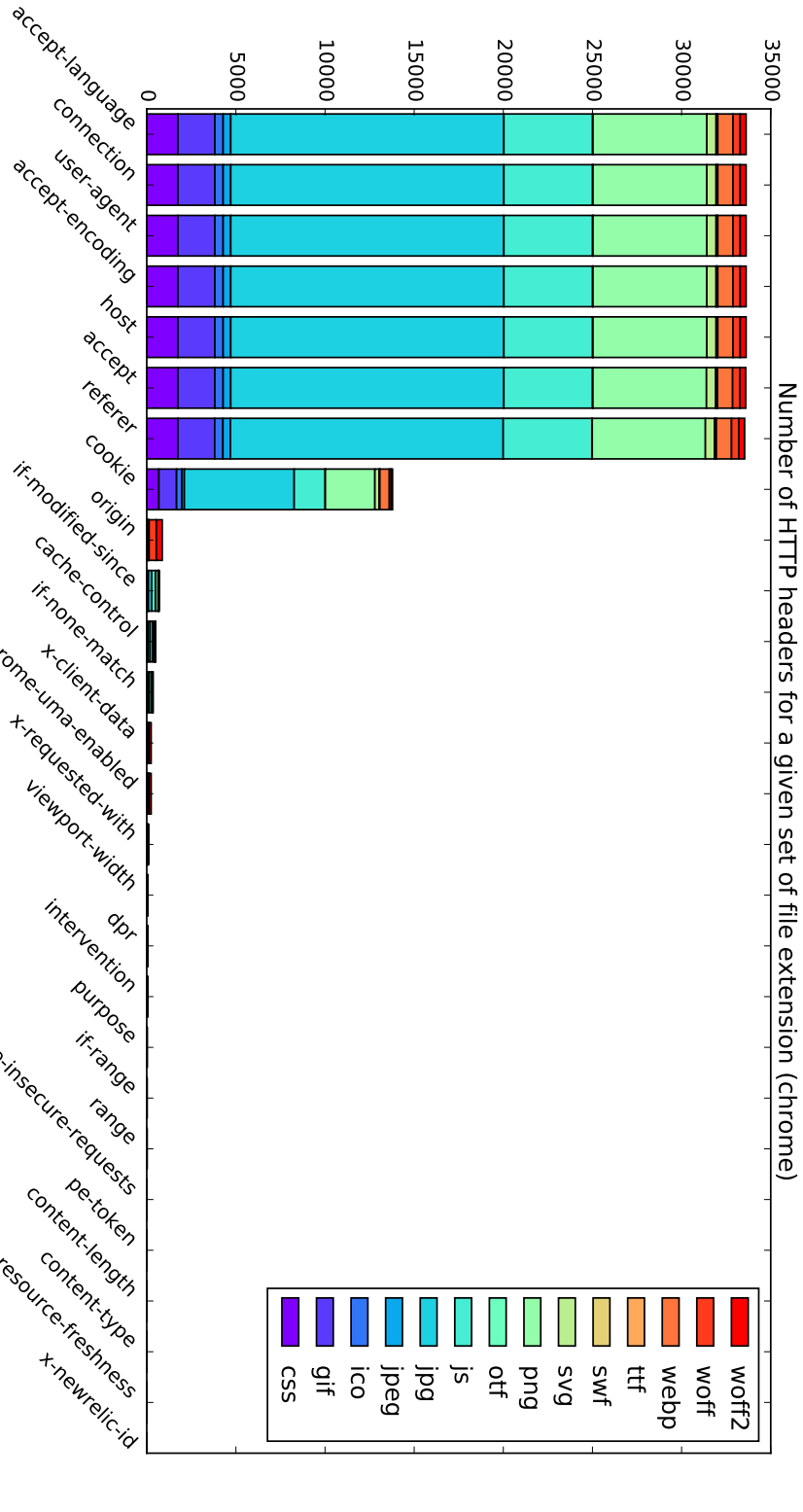


FIGURE 5.5 – Nombre d'occurrences de chaque champ de l'entête HTTP pour chaque extension de fichier avec Chrome

encoding, ce qui représente le plus petit ensemble de champs pour obtenir des requêtes similaires. *user-agent* décrit le navigateur utilisé ainsi que le système d'exploitation, cette valeur ne peut donc pas être la même entre Firefox et Chrome, *accept-language* décrit les langues préférées du navigateur avec des poids optionnels associés pour chaque langue et *accept-encoding* notifie au serveur les types d'encodage qui sont supportés par le navigateur. Vous pouvez retrouver les premières valeurs de quelques champs qui ont le plus d'impact sur la similarité des requêtes émises dans les Listings 5.1 and 5.2). En éditant ces trois champs nous obtenons 12% de similarité. Ce taux peut être fortement augmenté en ajoutant *accept* à la liste des champs à exclure et permet d'obtenir 51% de similarité, ce taux peut être encore augmenté en ajoutant le champ *cookie* et ainsi peut atteindre 80% de similarité. Les derniers champs *referer* et *connection* n'augmentent pas de manière significative ce taux avec un maximum de 83%.

Listing 5.1 – Valeurs les plus communes pour quatre des champs des requêtes envoyées par Firefox

```
accept-language:
  100.0% (34415) -- en-US,en;q=0.5

accept-encoding:
  98.0% (33720) -- gzip, deflate
  2.0% (695) -- identity

accept:
  90.2% (31045) -- */*
  5.1% (1765) -- text/css,*/*;q=0.1
  2.3% (806) -- application/font-woff2;q=1.0,application/font-woff;q=0.9,*/*;q
      =0.8
  2.2% (760) -- text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

user-agent:
  100.0% (34415) -- Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:50.0) Gecko
      /20100101 Firefox/50.0
```

Listing 5.2 – Valeurs les plus communes pour quatre des champs des requêtes envoyées par Chrome

```
accept-language:
  100.0% (33617) -- en-US,en;q=0.8

accept-encoding:
  100.0% (33614) -- gzip, deflate, sdch
  0.0% (3) -- gzip, deflate

accept:
  77.3% (25979) -- image/webp,image/*,*/*;q=0.8
  17.4% (5843) -- */*
  5.2% (1734) -- text/css,*/*;q=0.1
  0.1% (24) -- text/javascript, application/javascript, application/ecmascript,
      application/x-ecmascript, */*; q=0.01

user-agent:
  100.0% (33617) -- Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
      like Gecko) Chrome/55.0.2883.87 Safari/537.36
```

Nous avons conduit une dernière expérience pour évaluer à quel point nous pouvons ignorer certains des champs présents dans l'entête HTTP avant que la réponse du serveur ne commence a différer. Ainsi, nous évaluons le risque induit par les changements effectués que le serveur

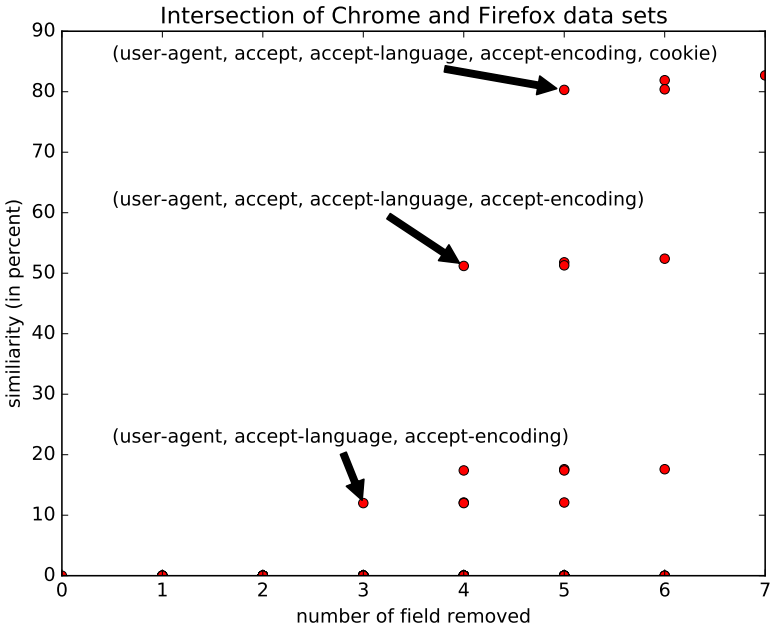


FIGURE 5.6 – Similarité entre le data set de requêtes HTTP de Firefox et Chrome avec différentes combinaisons de champs à exclure

produise une réponse différente de celle normalement attendue. Dans cette expérience, nous considérons les mêmes requêtes que celles présentes dans les data sets utilisés précédemment en les envoyant autant de fois que de combinaisons possibles avec les sept champs identifiés comme les plus importants. Cette expérience montre que la plupart des serveurs HTTP renvoient la même réponse peu importe les champs présents dans l'entête de la requête HTTP. Seulement une faible proportion de serveurs utilisent un encodage différent lorsque le client ne spécifie pas ceux qu'il est capable de comprendre dans le champ *accept-encoding*. Quelques autres, pour des ressources spécifiques, comme www.apple.com avec les fichier .woff, répondent avec le code HTTP 404 lorsque le champ *referer* n'est pas correctement renseigné (il doit contenir au moins le même domaine dans le cas de www.apple.com). De manière générale, nous sommes libres d'éditer les champs identifiés dans l'entête des requêtes pour améliorer les performances du cache du réseau NDN pour des contenus statiques sans conséquence sur l'exactitude des contenus.

Pour finir, les améliorations du taux de cache dues aux changements dans l'entête (en modifiant le champ *accept-encoding* et en ignorant les champs *user-agent*, *accept-language*, *accept* et *cookie* pour les contenus statiques uniquement) sont données sur la deuxième et troisième lignes de la Table 5.3. Nous mettons plus spécifiquement en évidence l'exclusion du champ *cookie* ou non car celui-ci est donné par le serveur pour par exemple identifier un utilisateur et dans ce cas il sera unique pour chacun. Cependant, il n'est pas nécessairement utile pour des ressources statiques sauf dans le cas où une politique de droit d'accès existe, mais cela est peu probable sur un site non chiffré. Lorsque le champ *cookie* n'est pas ignoré, on peut voir que pour deux utilisateurs utilisant un même navigateur, la proportion de contenus venant du cache ne change pas car les champs exclus contiennent déjà les mêmes valeurs car les navigateurs étant identiques, ils ont une configuration similaire. Cependant pour un utilisateur avec un navigateur différent, on peut voir que grâce à l'unification des requêtes HTTP, celui-ci est capable de récupérer des

contenus du cache alors que ce n'était pas le cas précédemment et ce à un niveau assez proche de notre autre utilisateur utilisant le même navigateur que notre référence avec 27,4% de cache hit contre 0% précédemmentt. En incluant *cookie* dans les champs à exclure, un nouvel utilisateur peut profiter du cache de la même façon que l'utilisateur ayant demandé en premier les contenus, que le navigateur soir le même (78,3%), ou de manière un peu moins prononcé avec un navigateur différent (61,3%).

5.4 Évaluation

5.4.1 Implémentation et environnement expérimental

Les *gateways* que nous proposons sont des composants purement logiciel implémentés en C++ et conçus sous la forme de Virtual Network Functions (VNF) qui peuvent être déployés où et quand cela est nécessaire. Nous pensons que le paradigme Network Function Virtualization [MSG+16] [HGJL15] peut faciliter le déploiement de NDN en réduisant les coûts nécessaires pour un fournisseur d'accès pour déployer et expérimenter ce type de réseau [MAM+16].

Les *gateways* peuvent gérer correctement les méthodes HTTP les plus courantes, à l'exception de la méthode HTTP CONNECT car cela créerait un tunnel TCP et l'architecture des *gateways* n'est pas conçue pour transmettre des flux de données illimités (un timeout peut survenir si aucune donnée n'est envoyée pendant un certain temps). Si l'on prend le cas de HTTPS, un client ne peut pas utiliser les *gateways* pour envoyer ce type de trafic mais cela ne fait pas partie des objectifs car transporter des paquets chiffrés de bout en bout avec TLS dans l'îlot NDN ne pourrait pas bénéficier des avantages de NDN (cache, multicast, etc.), et n'a donc aucun intérêt. Il faut savoir que les méthodes HTTP spécifiques aux API sont également acceptées, mais elles sont traitées comme les autres méthodes en raison de leurs similarités. Nous avons testé avec succès les méthodes HEAD, GET, POST et OPTIONS, tandis que PUT, DELETE, PATCH et TRACE ont également passé avec succès des tests unitaires plus simples. Pour utiliser les *gateways*, les clients web n'ont qu'à configurer l'adresse de la *gateway* d'entrée (iGW) comme proxy HTTP dans leur navigateur Web, le trafic HTTPS ne sera donc pas affecté et ne passera pas par l'îlot, ce qui fait sens.

Grâce à leur conception modulaire, comme précédemment illustrée sur la Figure 5.2, les composants des *gateways* peuvent être adaptés pour créer d'autres fonctions intéressantes. Par exemple, nous pouvons réutiliser le module de communication et l'« interpreter » de la *gateway* de sortie et les alimenter avec un moteur HTTP pour créer un serveur HTTP sur NDN. Une autre solution est de combiner la *gateway* de sortie avec un programme existant, par exemple, en la dédiant à un serveur Apache2 pour qu'il puisse aussi communiquer sur NDN. Le code des deux *gateways* est disponible en open-source [17] et un exemple de serveur web natif NDN est aussi disponible sous forme de preuve de concept.

Notre plateforme de test est composée de deux serveurs DELL PowerEdge R730. Chaque serveur dispose de deux processeurs Intel Xeon à huit cœurs à 2,4 GHz (E5-2630 v3) avec Hyper-Threading et Turbo Boost activé, 128 Go de RAM et deux 400 Go SAS SSD en RAID0 pour le système d'exploitation (serveur Ubuntu 16.04). Les serveurs sont directement interconnectés via des interfaces réseau Ethernet Intel 10Gbps (Intel X540) et sont connectés à Internet via une connexion symétrique 1Gbps. Chaque serveur exécute sa propre instance du routeur NDN (NFD 0.6.0) et utilise des *payloads* de 4096 octets au maximum. Nous avons réalisé nos expériences dans un réseau local qui implémente un réseau NDN construit autour d'un seul routeur NDN pour

17. https ://github.com/Kanemochi/NDN-HTTP-Gateway

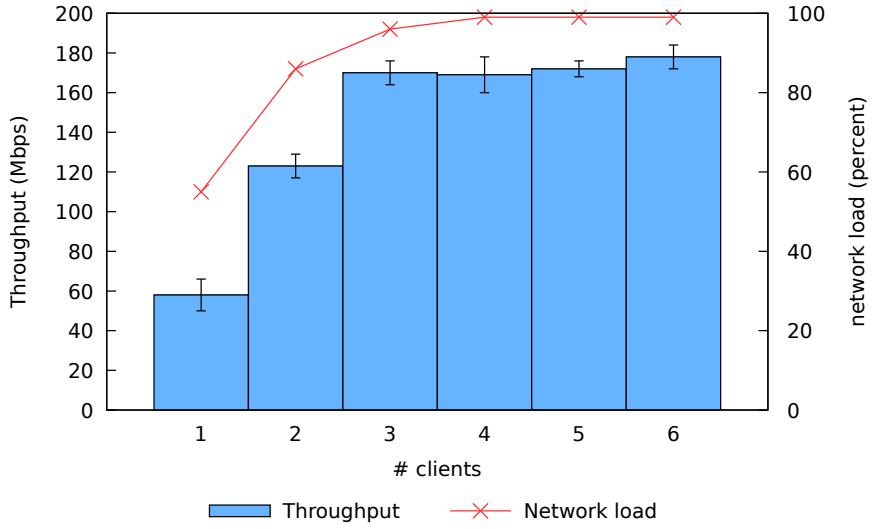


FIGURE 5.7 – Débit de l'îlot NDN et utilisation CPU du routeur NDN pour des transferts concurrents de fichiers

nos mesures, suivant la topologie illustrée sur la Figure 5.1. Le client web IP demande des objets HTTP via le réseau NDN et le serveur IP Apache2 local envoie les réponses correspondantes.

Dans les prochaines sous-sections, nous évaluons notre solution en terme de (1) performances synthétiques, (2) bénéfices pour les utilisateurs, (3) fiabilité et (4) gain attendu pour un fournisseur d'accès à Internet.

5.4.2 Tests de performances synthétiques

La Figure 5.7 montre les performances synthétiques de l'îlot NDN quand chaque utilisateur télécharge des fichiers différents et de grande taille via un lien Gigabit Ethernet. Cette expérience évalue le débit maximum que notre îlot est capable de fournir. Lorsque qu'un seul utilisateur demande un contenu, la *gateway* d'entrée est capable de délivrer un débit de 58 Mbps et avec trois utilisateurs en simultané, celle-ci peut délivrer jusqu'à 172 Mbps, valeur qui est limitée par l'utilisation CPU du routeur NDN, donné par la courbe « NFD CPU load » sur la Figure 5.7. Cette limitation du routeur a déjà été investiguée dans le chapitre 3.

La Figure 5.8 montre le délai additionnel induit par l'usage des *gateways*. Pour cette expérience, nous avons décidé de limiter la vitesse du lien de l'utilisateur à 10 Mbps pour que seulement le délai additionnel affecte le temps de téléchargement, car comme vu plus tôt, un utilisateur ne peut obtenir que 58 Mbps au maximum de la part des *gateways*. Les résultats mettent en évidence que le délai additionnel est stable avec environ 29±3ms et surtout constant peu importe la taille du fichier téléchargé. Comme expliqué dans la précédente section, ce délai constant est rendu possible grâce au fait que les *gateways* n'attendent pas de récupérer la totalité des messages HTTP avant de commencer le transfert des *chunks* (paquets NDN *Data*) mais commencent dès qu'un entête est valide, streamant ainsi le flux des données.

Le dernier test de performances synthétique que nous avons effectué cherche à estimer les effets du cache du réseau NDN lorsque le serveur HTTP est distant. Ainsi dans cette expérience, nous appliquons un délai réseau de 50ms sur le serveur HTTP et tous les liens utilisés ont une vitesse de 1Gbps. Il n'y a pas de délai additionnel entre les utilisateurs et l'îlot NDN car celui-ci est supposé être proche des utilisateurs. La Figure 5.9 nous donne les résultats de cette expérience

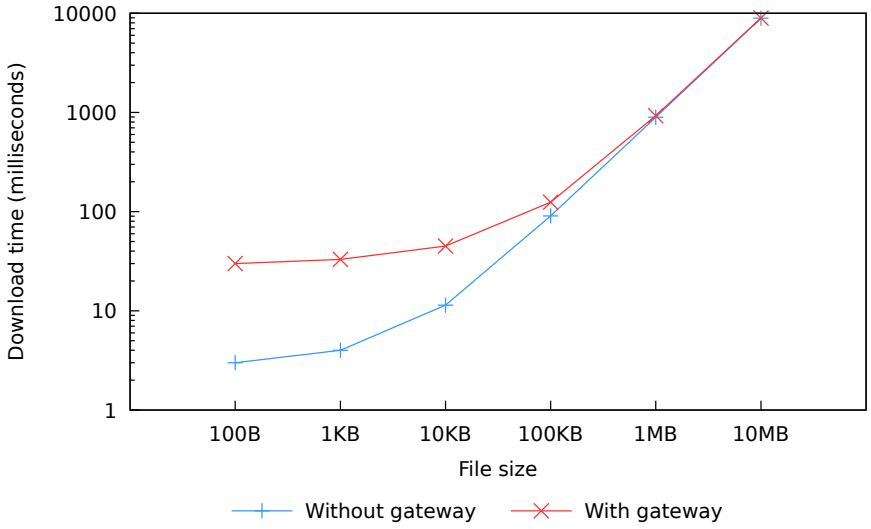


FIGURE 5.8 – Temps nécessaire pour un client pour télécharger un fichier avec un lien de 10 Mbps avec et sans les *gateways*

pour le téléchargement d'une image de 100KB selon différentes conditions. Sans cache, il est plus rapide de récupérer l'image directement sans passer par l'îlot NDN. Les temps restent assez similaires dans le cas d'une réponse HTTP 200 (le contenu) avec 255 ms sans les *gateways* et 281 ms avec, mais l'est beaucoup plus dans le cas d'une réponse HTTP 304 (signifiant qu'il n'y a pas eu de changement avec la version précédente) avec 102 ms sans les *gateways* et 154 ms avec. Il est totalement normal que la solution des *gateways* soit moins performante lorsque le cache n'est pas sollicité car les *gateways* ajoute un délai supplémentaire pour chaque requête, mais comme vu dans la précédente expérience, ce délai devient moins perceptible plus le fichier est volumineux car ce délai reste constant. Cela explique aussi pourquoi la différence est plus marquée pour la réponse HTTP 304 car celle-ci ne contient qu'un entête contrairement à la réponse HTTP 200 qui contient aussi le contenu. Cependant, si le cache contient déjà l'image demandée, notre solution devient plus performante car elle permet de ne pas subir l'éloignement du serveur HTTP car la réponse en cache se trouvera « proche » de l'utilisateur et ainsi les délais pour récupérer les réponses HTTP 200 et 304 sont respectivement de 15 ms et 4 ms.

5.4.3 Bénéfice pour les utilisateurs finaux

Dans cette sous-section, nous utilisons un outil développé par Orange, nommé MORIS (Measuring and Observing Representative Information of webSites). Les mesures ont été réalisées par Antoine Saverimoutou, doctorant chez Orange Labs, dans le cadre du projet DOCTOR. MORIS permet d'automatiser des tests de navigation et, pour chaque site visité, de récupérer des métriques comme le temps de chargement de la page (PLT [18]) pour avoir une bonne estimation de la qualité d'expérience (QoE) ressentie par l'utilisateur. L'outil MORIS récupère aussi différents indicateurs comme le nombre d'éléments qui composent une page web, la localisation de leur serveur, l'impact sur le réseau, etc... Toutes ces données collectées par l'outil MORIS peuvent aider ses utilisateurs à analyser les performances des pages web demandées et identifier les paramètres qui jouent le plus sur la QoE.

18. https ://www.maxcdn.com/one/visual-glossary/page-load-time/

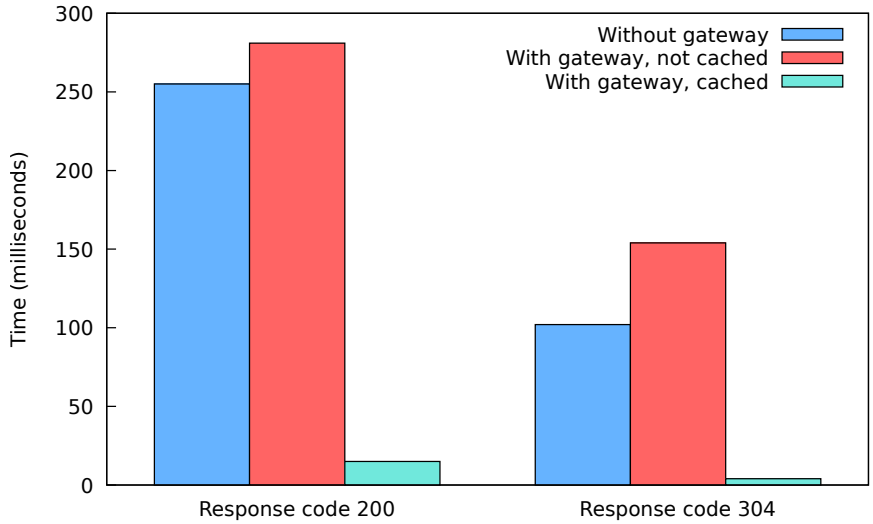


FIGURE 5.9 – Effet du cache NDN sur le temps de téléchargement d'une image PNG de 100 KBytes

TABLE 5.4 – Taux d'utilisation de HTTP et HTTPS pour les sites web les plus populaires (en % du Top 10000)

	NA	SA	EU	AS	AF	OC
Full HTTP	14,95	0,16	9,12	4,7	0,13	0,1
Mixed	2,96	0,06	2,04	1,02	0,02	0.04
Full HTTPS	34,01	0,37	17,59	6,07	0,13	0.11

Proportion de trafic redirigé

Le trafic web actuel va devenir de plus en plus chiffré, ainsi l'intérêt actuel d'une *gateway* intermédiaire pour convertir le trafic HTTP sur NDN pourrait être remis en question. Nous voulons évaluer ici l'intérêt de notre solution pour l'écosystème web actuel. Pour cela, avec l'outil MORIS, nous avons effectué des tests sur les sites répertoriés comme les plus populaires dans le Top 10000 d'Alexa pour savoir s'ils délivrent des contenus en HTTP ou HTTPS, classés par continents. Dans la Table 5.4, nous donnons les pourcentages des sites accédés via HTTP ou HTTPS (page d'accueil) présent dans le Top 10000 d'Alexa et dans la Table 5.5, le pourcentage de ressources récupérées via HTTP ou HTTPS pour ces 10000 sites car une page web est composée de plusieurs ressources qui ne sont pas nécessairement accédées avec le même protocole. Ces deux Tables montrent que même si HTTPS est le protocole dominant, HTTP représente toujours une part significative des sites web les plus populaires avec environ un tiers du trafic, et pour certains continents, la proportion de ressources HTTP est même plus importante que celle en HTTPS. La Figure 5.10 illustre la distribution pour les 1000 premiers sites les plus populaires. On peut donc conclure que nos *gateways* peuvent encore avoir un intérêt pour le trafic web. À moyen terme, on peut entrevoir une architecture mixte où les contenus publics (ne comportant pas d'information personnelle comme par exemple des vidéos Netflix ou Youtube) sont délivrés via NDN nativement ou à l'aide de *gateways* HTTP/NDN comme les nôtres alors que les informations privées resteraient sur HTTPS/IP.

TABLE 5.5 – Taux d'utilisation de HTTP et HTTPS pour les ressources des sites web populaires (en % of du Top 10000)

	NA	SA	EU	AS	AF	OC
HTTP	16,32	0,53	10,72	8,12	0,19	0,12
HTTPS	35,27	0,59	20,11	7,56	0,15	0.12

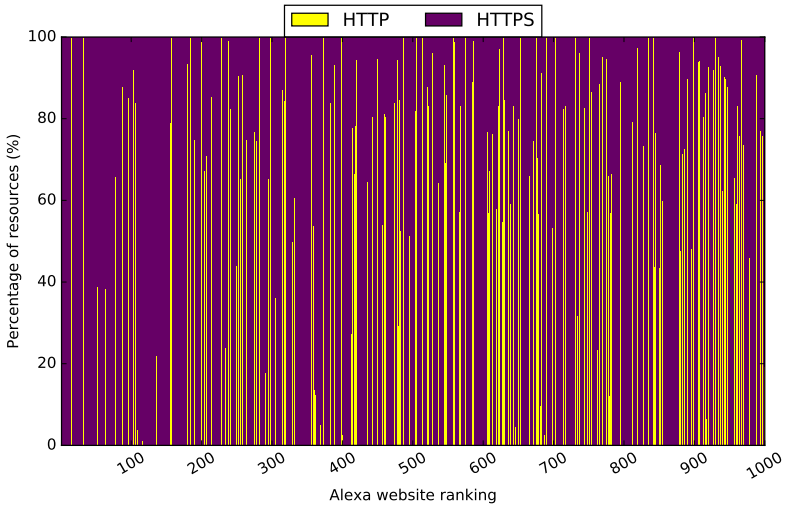


FIGURE 5.10 – Répartition des ressources HTTP et HTTPS pour les 1000 premiers sites web

Qualité perçue par les utilisateurs finaux

Nous avons évalué le temps de chargement des pages d'accueil des sites du Top 10000 Alexa. Chacun de ces sites peut être composé de plusieurs ressources HTTP et/ou HTTPS. Nous avons fait ces expériences en utilisant le réseau Internet traditionnel (IP), en utilisant les *gateway* une première fois (sans cache) ainsi qu'une seconde fois pour voir les avantages procurés par le cache du réseau NDN. Cette expérience nous permet d'estimer la qualité que peuvent percevoir les utilisateurs en utilisant nos *gateways* en la comparant à la situation actuelle. Pour obtenir des valeurs représentatives, les tests sont effectués avec l'outils MORIS connecté grâce à une ligne FTTH d'Orange. Concernant l'îlot NDN, nous utilisons la même topologie, serveurs et liens (1Gbps) que la sous-section précédente.

Les Figures 5.11 et 5.12 montrent que lors d'une première visite l'utilisation des *gateways* induit une augmentation du temps de chargement des pages d'accueil et il est d'autant plus important que le serveur est distant de l'îlot NDN. Cela peut s'expliquer en partie car la *gateway* de sortie ouvre de nouvelles connexions pour chaque requête et doit ainsi refaire le *handshake* TCP à chaque fois mais aussi par le fait que l'usage des *gateways* implique un temps additionnel fixe pour la conversion des messages HTTP comme vu précédemment, auquel s'ajoute la distance qui sépare l'îlot NDN du client (le client est à Lannion et l'îlot NDN à Nancy, soit environ 700 kilomètres en ligne droite). Pour la seconde passe, les résultats sont bien meilleurs que les premiers. Les valeurs médianes (Figure 5.12) sont au moins égales à celles sans l'utilisation des *gateways*, voire parfois meilleures pour les serveurs les plus lointains (Afrique du Sud, Asie, Océanie). Globalement, les sites web en Europe ou en Amérique du nord sont suffisamment rapides pour que les effets du cache NDN ne compensent pas le délai induit par l'usage des

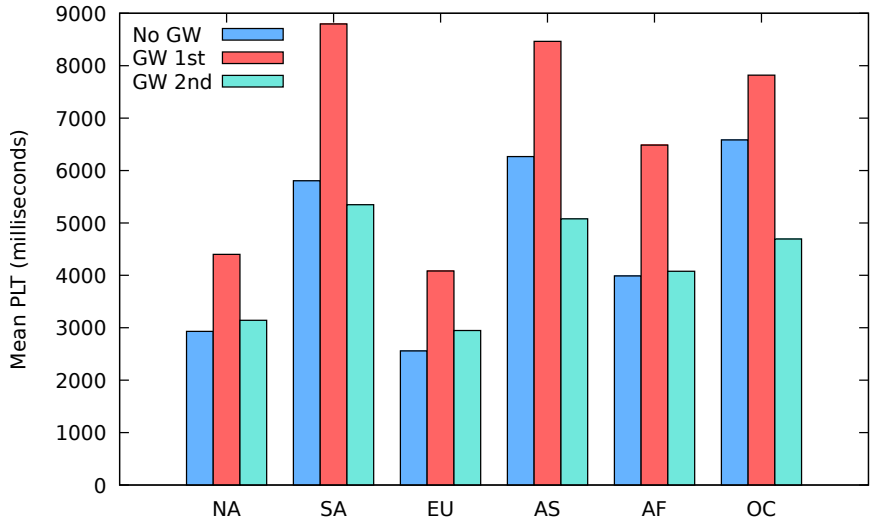


FIGURE 5.11 – Valeurs moyennes du temps de chargement des pages d'accueil des sites web les plus populaires en utilisant ou non les *gateways*

gateways.

5.4.4 Fiabilité

Nous voulons évaluer si les *gateways* peuvent efficacement mapper les requêtes et réponses HTTP et les transporter dans le réseau NDN de façon fiable (sans perte de paquets ou durée de transit excessive conduisant à l'expiration de timers pour les entités HTTP). Nos expériences consistent à charger via notre îlot les pages d'accueil et tous leurs éléments des 1000 sites HTTP les plus populaires selon Alexa. Nos résultats sont basés sur un scraper web construit autour de l'API JAUNT et développé par Moustapha El Aoun, un ingénieur de l'Université Technologique de Troyes. Cet outil fournit des métriques permettant d'identifier les causes d'une erreur HTTP (problème de connexion, de serveur, ou incapacité des *gateways* à fournir le contenu).

En tout, nous avons collecté environ 38000 objets HTTP. La Figure 5.13 montre la répartition des pages d'accueil selon le nombre d'objets qu'elles contiennent ainsi que le taux d'erreur pour chaque classe. On peut voir que le taux d'erreur reste faible (globalement $< 1\%$) et que les pages avec peu d'objets ont une probabilité plus forte d'obtenir une erreur (jusqu'à 3.8%) pour ceux avec moins de dix éléments (une erreur a plus d'impact sur un site contenant peu d'éléments). Le taux d'erreur inclut toutes les requêtes qui ont expiré ainsi que les erreurs côté client (codes HTTP en 4xx) qui ne sont pas clairement identifiées comme venant du scraper. Nous gardons ces erreurs, notamment les code 404, car elles peuvent aussi bien venir du scraper que des *gateways* à cause d'un changement dans l'entête ou d'une ressource temporaire.

Nous avons aussi tracé sur la Figure 5.14 la distribution de ces sites HTTP selon le nombre d'objets correctement récupérés. La grande majorité des pages d'accueil des sites visités sont (quasi) parfaitement récupérées et seulement une faible proportion (19 sites sur les 1000 visités) montrent de mauvais résultats (moins de 75% d'éléments récupérés correctement). Cette catégorie de sites est essentiellement composée de sites web lointains (par exemple chinois), ainsi le problème pourrait venir de délais trop long causant des timeouts au niveau des *gateways*.

Parmi les objets collectés, dont les nombres sont donnés sur la Figure 5.15, 70,88% sont des images ou du moins des contenus présents dans des balises HTML « *img* », 15.29% sont des

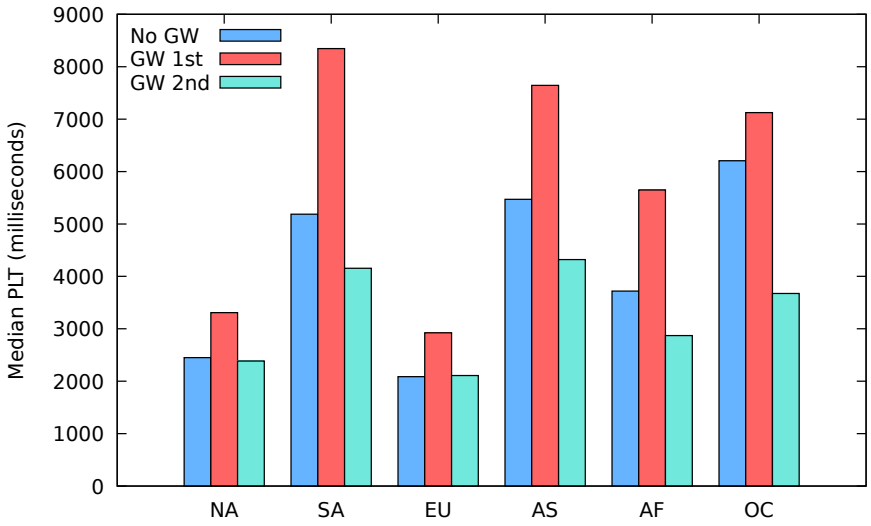


FIGURE 5.12 – Valeurs médianes du temps de chargement des pages d'accueil des sites web les plus populaires en utilisant ou non les *gateways*

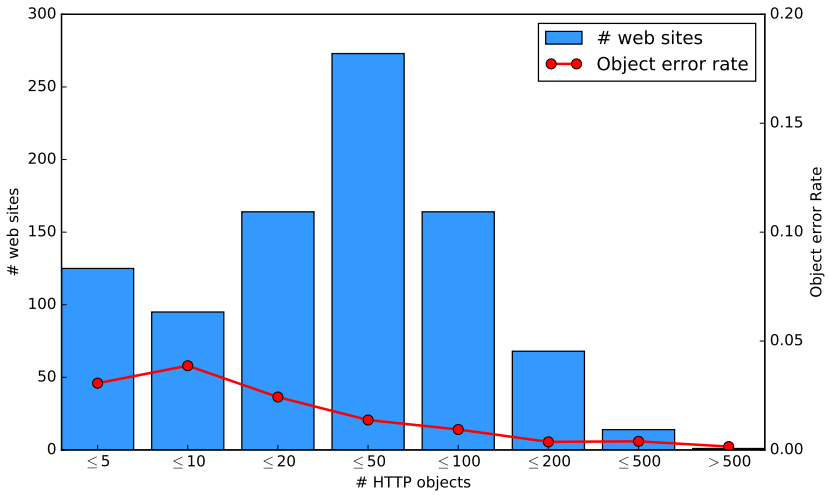


FIGURE 5.13 – Nombre de sites webs mesurés et leur taux d'erreur en fonction du nombre d'objets HTTP par page

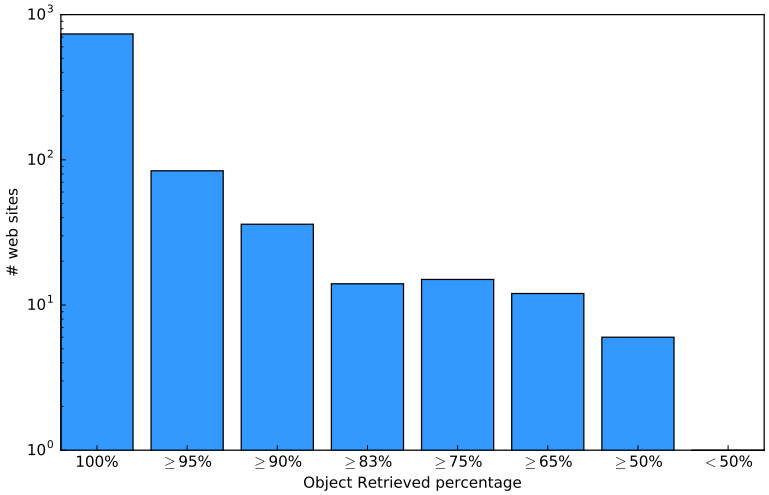


FIGURE 5.14 – Distribution des 1000 premiers sites HTTP selon le taux d'éléments récupérés

ressources externes (appelées via des balise « *link* »), 12,88% sont des scripts et les derniers 0.95% sont des ressources hétérogènes comme des *iframe*, etc... De manière générale, le taux d'erreur pour chaque « type » de contenu est négligeable sauf pour les *iframe* et dans une moindre mesure les ressources externe (*link*) avec respectivement (2,5% et 1,3%).

Pour mieux caractériser l'origine de ces erreurs, nous avons tracé sur la Figure 5.16 le taux d'erreur selon le TLD utilisé (Top Level Domain). On peut ainsi voir que les ressources présentent sous certains TLD ont plus de chance d'obtenir une erreur que d'autres. Les cinq TLD qui ont le plus haut taux d'erreur sont dans l'ordre : .pt (Portugal) et .it (Italie) avec 3.3% et 3.1%, suivie de .tw (Taïwan) et .ir (Iran) avec 2.5% et 2%, puis de .tr (Turquie) avec 1.7%. Mais comme la majorité des ressources viennent de domaines en .com ou .net, nous ne pouvons pas tirer de conclusion précise, mais seulement des indices sur quels domaines les erreurs ont le plus de chance de se produire.

5.4.5 Gain attendu grâce au cache du réseau NDN

Comme dernière expérience, nous avons voulu savoir dans quelle mesure un fournisseur d'accès à Internet peut économiser du trafic grâce à l'utilisation d'un réseau NDN à la place d'IP pour des contenus HTTP. Dans cette expérience, nous essayons de simuler le comportement d'utilisateurs à l'aide de Selenium et de quatre navigateurs marionnettes Firefox. Ces marionnettes suivent les règles suivantes : (1) un site web est sélectionné selon une loi Zipf de probabilité simulant la popularité, (2) après avoir récupéré la page d'accueil (défini par 'return document.readyState' == 'complete), la marionnette va attendre quelques secondes en se basant sur une loi de poisson avec un paramètre fixé à 4 avant de (3) sélectionner la prochaine ressource à visiter, parmi celles présentes dans les balises HTML '*a*' (hyperlien) de la page selon une loi uniforme. Nous avons défini le nombre de liens à visiter suivant une courbe de vieillissement négative comme présenté dans [LWD10] mais étendu à tout le site web. De plus, le routeur NDN a une taille de cache définie à environ 1GB (2^{18} paquets de 4KB maximums). La Figure 5.17 montre l'évolution du taux de données (paquets, à cause de sa granularité) réutilisées du cache du routeur NDN. Ainsi, plus le temps passe, plus la courbe devrait tendre vers sa valeur moyenne. La Figure 5.17 donne les courbes pour des loi Zipf de paramètre 1,2 et 1,5.

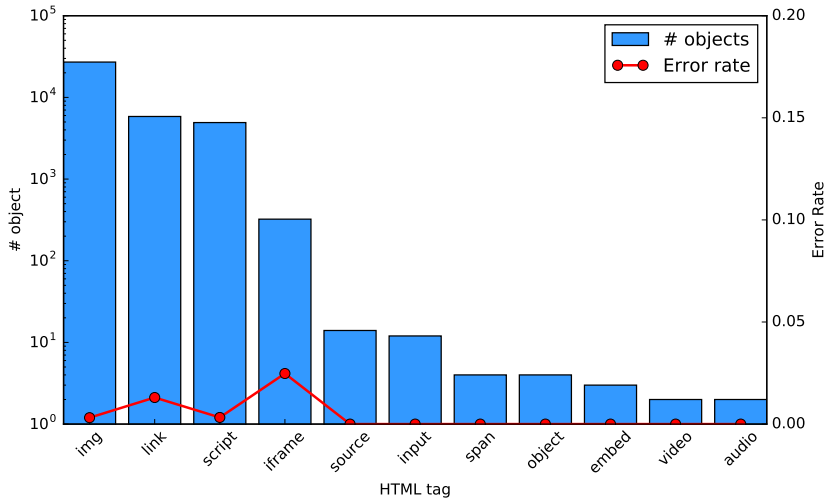


FIGURE 5.15 – Nombre d'objets recenséset leur taux d'erreur en fonction du type d'objet HTML

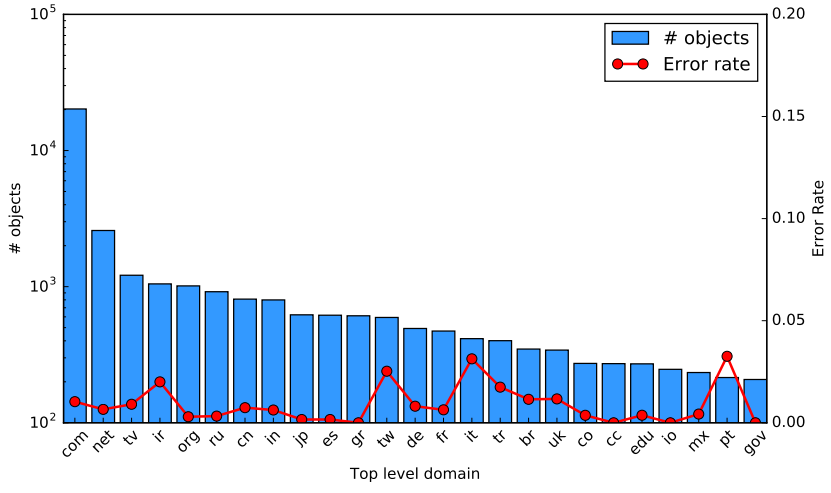


FIGURE 5.16 – Nombre d'objets recensés et leur taux d'erreur en fonction du top-level domain

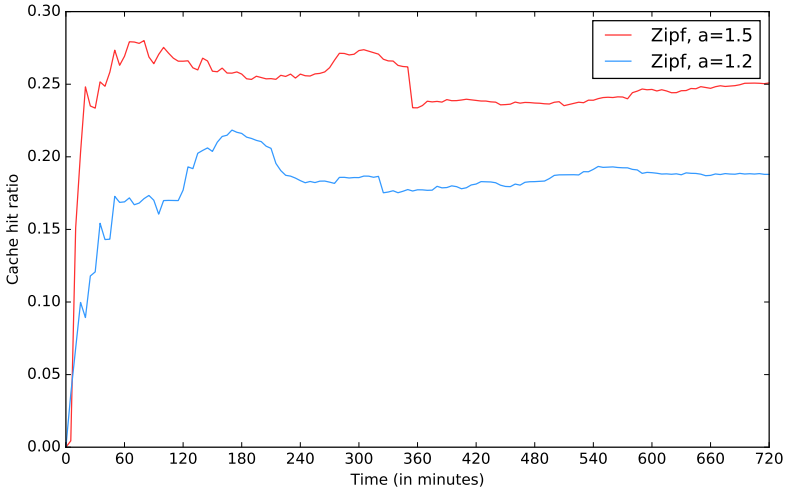


FIGURE 5.17 – Évolution du taux de cache du réseau NDN dans le temps

Comme supposé, la diversité de la popularité des sites web visités a un impact notable sur le taux de cache des contenus : plus le navigateur se concentre sur les sites populaires (paramètre de la loi Zipf élevé) plus le cache est efficace. On peut apercevoir quelques chutes du taux de cache assez localisées, celles-ci sont majoritairement dues à des téléchargements de fichiers binaires assez volumineux qui ne seront jamais réutilisés ou qui ont une directive interdisant leur mise en cache. Après plusieurs heures de fonctionnement, la valeur du taux de cache s'élève à 18,8% et 25,1% respectivement pour un paramètre de 1,2 et 1,5 pour la loi Zipf.

Bien qu'ils soient prometteurs, parce qu'ils prédisent une économie de bande passante substantielle, ces chiffres sont également difficiles à interpréter dans un contexte plus large. Ils seront affectés par la taille des caches utilisés par les opérateurs et les habitudes de leurs utilisateurs. Une autre question importante est de savoir si les contenus vidéo, qui bénéficieraient grandement de la mise en cache, resteraient sur HTTPS dans le cas où les réseaux ICN seraient largement déployés. Ces résultats sont comme un verre à moitié vide ou à moitié plein selon la façon dont nous considérons le protocole NDN. S'il est considéré comme un nouveau protocole de transport (comme TCP), la mise en cache au niveau des paquets est toujours bénéfique. Si l'on considère NDN comme une architecture de diffusion de contenu (comme un CDN), le faible taux d'accès au cache et l'absence d'un cache considérant les contenus dans leurs globalité font qu'il est difficile pour le protocole NDN de concurrencer les CDN actuels.

5.5 Conclusion

Notre objectif était de concevoir et d'évaluer une solution permettant de transporter efficacement le protocole HTTP sur NDN afin de rendre le protocole NDN interopérable avec HTTP, le principal protocole actuel de diffusion de contenu sur Internet, et ainsi de répondre au besoin de la communauté NDN de travailler avec du trafic réel. Dans ce chapitre nos contributions ont été les suivantes :

— Un protocole de mappage et une architecture pour transporter du trafic HTTP hétérogène sur NDN ;

— Une optimisation pour mutualiser les requêtes HTTP des utilisateurs pour mieux bénéfi-

cier des fonctionnalités de NDN ;
— Des *gateways* opérationnelles et dont le code est open-source ;
— Une évaluation approfondie de ces *gateways* ainsi que le gain espéré pour un fournisseur d'accès de transporter HTTP sur NDN.

Nous proposons ainsi un protocole permettant de transformer et transférer des messages HTTP provenant du monde IP sur NDN et vice versa, tout en essayant de respecter le plus possible les spécificités de l'architecture NDN afin de tirer parti de ses avantages. De plus, notre protocole n'est pas limité à la conversion entre deux *gateways* et peux donc être utilisé par des utilisateurs et fournisseurs de contenus web natifs NDN. Nous avons montré que la traduction de HTTP sur NDN n'est pas simple et que notre protocole de mappage, défini par le schéma de communication entre les composants et le schéma de nommage suivi par les messages échangés, doit être soigneusement conçu pour respecter le protocole NDN et permettre un transport efficace de HTTP. Ceci est rendu possible par une unification efficace des requêtes HTTP entre les utilisateurs pour bénéficier des propriétés natives de cache et de diffusion de masse du protocole NDN. Notre solution est implémentée sous forme de *gateways* qui remplissent les objectifs fonctionnels initiaux affichés : être transparent pour les utilisateurs finaux, adaptatif pour permettre la communication entre les mondes NDN et IP quelques soient les configurations, efficace le transport des contenus web et, pour finir, fiable. Les performances des *gatewyas* ont été étudiées en profondeur autour de 4 axes : les performances synthétiques, le bénéfice pour les utilisateurs finaux, la fiabilité et le gain attendu de l'utilisation du cache du réseau. Les résultats montrent que les *gateways* offrent de bonnes performances et sont fiables. Du côté de l'utilisateur, le surcoût est très raisonnable, en particulier pour les contenus populaires présents dans le cache, alors que du côté des FAI, le taux d'utilisation du cache prédit une précieuse économie de bande passante, mais pas au point de faire du protocole NDN une concurrence sérieuse aux solutions à base de CDN. Nos évaluations nous amènent à conclure que les *gateways* sont pleinement opérationnelles. Le code source est d'ailleurs mis à disposition de la communauté.

Les gateways que nous avons conçues permettent de créer des îlots NDN pouvant être déployés progressivement où et quand nécessaire pour aider à la diffusion de certains contenus. Ce déploiement progressif est aujourd'hui rendu possible par la virtualisation des réseaux qui permet à plusieurs protocoles de même niveau de cohabiter en isolation. Dans le prochain chapitre, nous proposons d'appliquer le paradigme NFV au réseau NDN. En particulier, nous avons conçu une solution à base de microservices orchestrés permettant de déployer facilement puis d'administrer un îlot NDN opérationnel dont les *gateways* sont l'un des composants clés.

Chapitre 6

Proposition d'une architecture à base de microservices pour le protocole NDN

Sommaire

6.1 Introduction

Depuis peu, il existe un engouement pour la *softwarization* des réseaux, en particulier grâce au paradigme de virtualisation des fonctions réseaux ou *Network Function Virtualization* (NFV) [Gro13]. Ce paradigme pose les bases de standards qui vont grandement faciliter la mise en place de fonctions réseaux virtualisées aussi nommées *Virtualized Network Functions* (VNF) sur des serveurs x86 et ainsi remplacer les équipements des infrastructures réseaux actuelles qui sont

souvent onéreux et spécifiques. NFV apparaît comme une opportunité pour des protocoles comme NDN car celui-ci pourrait ainsi être déployé incrémentalement à côté d'autres protocoles mais aussi à moindre coût grâce aux infrastructures NFV. Cela pourrait lever les principaux verrous qui limitent actuellement l'adoption des ICN et ainsi permettre un déploiement progressif des ICN par les fournisseurs d'accès quand et où ils sont nécessaires.

NFV permet également l'application de patrons de conception hérités du monde logiciel à celui des réseaux informatiques. Parmi eux, les architectures basées sur les microservices ont rapidement émergé et semblent prometteuses pour résoudre des problèmes comme les performances et mise à l'échelle des fonctions réseaux en donnant plus d'opportunités pour les déployer de manière efficace. Par exemple, dans les réseaux ICN, il est reconnu que les *Content Stores* ne sont pas utiles sur tous les nœuds mais à des endroits spécifiques sur le réseau [SFG+14] [FLT+13]. Cela permettrait aussi d'isoler la *Pending Interest Table* (PIT), élément *stateful* d'un routeur NDN, dans un microservice ce qui facilitera la mise à l'échelle des autres éléments car son impact sur ceux-ci sera limité en comparaison avec un routeur monolithique. Ainsi, nous attendons plusieurs avantages d'exécuter NDN en tant qu'architecture à base de microservices virtualisés par rapport à une implémentation monolithique, en particulier :

— un déploiement incrémental de NDN sur un environnement NFV existant ;
— des topologies NDN plus efficaces ;
— une meilleure mise à l'échelle horizontale pour obtenir de meilleures performances lorsque cela est nécessaire ;
— une capacité du réseau à déployer dynamiquement de nouvelles fonctions, comme des fonctions de sécurité, là où elles sont nécessaires ;
— un développement plus simple et séparé pour chaque module (moins de couplage et de complexité).

Néanmoins, les architectures à base de microservices doivent faire face à de nombreux défis. Même si les services sont plus simples, cela se fait au détriment d'une certaine complexité de chaînage mais aussi d'une orchestration des microservices pour obtenir des réseaux cohérents. Le plein potentiel d'une architecture à base de microservices, qui se base fortement sur la reconfiguration dynamique du réseau, ne peut être atteint que si l'architecture est correctement orchestrée. C'est pourquoi, pour révéler les propriétés souhaitables des microservices, nous avons également conçu et mis en œuvre un plan de management complet.

Dans ce chapitre, nous proposons la première réalisation du routage du protocole NDN en utilisant une architecture à base de microservices, qui englobe plusieurs contributions :

— La conception de sept microservices pour répondre aux exigences NDN et assurer de bonne performances et la sécurité du réseau ;
— Des exemples de combinaisons possibles pour réaliser des topologies de réseau NDN efficaces ;
— Une évaluation de l'ensemble de l'architecture selon deux scénarios mettant en cause les performances et la sécurité du réseau ;
— L'implémentation open source de l'architecture proposée.

La suite de ce chapitre est organisée comme suit. La section 6.2 introduit les architectures à base de microservices. La section 6.3 décrit les sept microservices développés qui composent notre architecture et comment ils peuvent être assemblés. Ensuite, nous expliquons comment ils sont gérés et orchestrés dans la section 6.4. La section 6.5 présente notre implémentation et la section 6.6 l'évaluation de notre architecture sous différents scénarios. Enfin, la section 6.7 traite des leçons à tirer et la section 6.8 conclut le chapitre.

6.2 Les architectures à base de microservices appliquées aux réseaux

Les premiers à avoir décrit les architectures à base de microservices sont J. Lewis et M. Fowler [Fow14]. Cette architecture logicielle est une façon de découper une application monolithique en un ensemble de petits programmes n'ayant qu'une seule tâche précise à exécuter (le service) et étant aussi capable de communiquer avec les autres services à l'aide d'un protocole commun à tous (généralement une API HTTP REST). L'architecture à base de microservices peut être vue comme une extension de l'architecture orientée service (SOA) [KSG+17], mais avec l'ajout de concepts supplémentaires, comme la communication directe entre services. Les microservices sont actuellement majoritairement utilisés pour les applications *Cloud* et l'IoT (*internet of things*).

Une grande partie de la littérature, comme [Ric17] [DM14], reconnaît de nombreux avantages aux microservices, tels qu'un développement et une maintenance plus simple par rapport à son équivalent monolithique, avec par exemple une équipe de développement dédiée par microservice. Cela est dû au fait qu'au cours du temps, une application monolithique va grossir et devenir de plus en plus complexe avec l'ajout de nouvelles fonctionnalités au point d'être complexe et difficilement appréhendable pour un nouvel arrivant. En outre, cette complexité et le code historique peuvent aussi bloquer les technologies utilisées à cause des changements trop profonds (un bon exemple est les applications bancaires dont certaines sont encore en COBOL). Les microservices, en restant simples, n'ont pas ce problème et permettent ainsi d'être plus réactif aux nouvelles technologies. De plus, de par la relative simplicité de chaque service, une application à base de microservices peut être plus facilement mise à l'échelle car seuls les éléments limitants auront besoin d'être répliqués au lieu de l'application complète. Mais les microservices n'offrent pas que des avantages, parmi les inconvénients les plus importants, on retrouve par exemple, un déploiement de l'application plus complexe et la nécessité d'implémenter un protocole de communication entre les microservices pour les composer. D'une façon générale, les microservices ont tendances à consommer plus de ressources que leur équivalent monolithique, dû par exemple à la réplication de données ou l'encodage/décodage des messages. Ueda et al. [UNO16] vont dans ce sens, leurs résultats sur l'étude d'une application web existant sous forme monolithique et microservices mais aussi codée avec deux langages (Java et node.js) pour chacune des deux architectures, ils montrent qu'utiliser une architecture à base de microservices peut augmenter le surcoût en performance jusqu'à 79.2% à cause de changements de contexte plus fréquents et une moins bonne utilisation du cache au niveau du CPU. De plus, les microservices ont tendance à être utilisés dans des conteneurs (virtualisation de type 0), ce qui peut encore amplifier ce surcoût. Les auteurs montrent que l'utilisation d'un réseau virtualisé, un bridge Docker dans leur cas, peut réduire les débits jusqu'à 33.8%.

Les microservices ne sont pas limités aux applications *Cloud* et IoT mais peuvent être utilisés pour n'importe quel type d'application et une partie de la recherche tente de créer des outils pour aider les développeurs à passer d'une application monolithique aux microservices. Par exemple, Levcovitz et al. [LTV16] proposent une méthodologie, basée sur un graphe de dépendance, pour identifier les éléments facilement isolables dans une application monolithique. Ils ont ensuite validé cette méthodologie sur une véritable application interne bancaire. Avec l'avènement de la *softwarization* des applications réseaux et d'initiative comme *ClickOS* [MAR+14] ou *Unikernel* [MMR+13], mais aussi grâce à de nouveaux standards comme NFV de l'ETSI, les fonctions réseau tendent à suivre la même voie vers la création d'architectures réseau évolutives et robustes à partir de petites VNF orchestrées avec la promesse pour les opérateurs réseau de favoriser l'innovation tout en réduisant leurs coûts d'investissement et d'exploitation (OPEX et CAPEX).

Si l'on reste dans le domaine des ICN, des initiatives tentent d'améliorer le traitement du réseau grâce au traitement de l'information sur le trajet dans un réseau ICN. Les auteurs de [SKST14] envisagent la mise en place de fonctions nommées sur un réseau CCN et proposent de distribuer les fonctions de traitement sur le réseau en ajoutant un *resolver* dans les nœuds CCN pour interpréter des expressions λ intégrées dans les noms NDN et qui peuvent être identifiés avec un préfixe spécifique. De façon similaire, le framework NFaaS [KP17] (Named Function as a Service) utilise des techniques basées sur le *Cloud* pour déployer des machines virtuelles légères basées sur *Unikernels* [MMR+13] pour une exécution à la demande des services sur les nœuds du réseau. Les machines virtuelles sont stockées dans un composant, nommée *Kernel Store*, qui peut également gérer l'exécution des machines virtuelles afin qu'un service puisse migrer entre les nœuds dans le temps. Ces deux travaux ont pour but d'exécuter des fonctions de traitement de l'information sur les nœuds du réseau tandis que nos microservices sont entièrement dédiés aux tâches réseau. Toutefois, comme ils opèrent à un niveau différent, il devrait être possible et fructueux d'utiliser les deux approches en même temps.

En soit, l'architecture à base de microservices est un concept prometteur mais elle n'a pas encore été réellement appliqué au domaine des réseaux. Dans le même temps, il y a une réelle tendance vers l'utilisation des réseaux « logiciels » (à base de programmes et non d'équipements) grâce aux paradigmes SDN et NFV pour le déploiement de nouveaux réseaux comme les ICN mais la majorité des propositions utilisent des éléments monolithiques et prêts à l'usage, ce qui limite les options et la flexibilité de ces réseaux virtualisés. Ainsi, nous nous donnons le défi de proposer une architecture à base de microservices spécifiquement développée pour les ICN et permettant de mieux tirer parti des propriétés des ICN et de technologies comme NFV que l'on nommera μNDN le long de ce chapitre.

6.3 Un ensemble de microservices pour le protocole NDN

Dans cette section, nous décrivons les différents microservices que nous avons développés sous forme de VNF. Nous les avons divisés en deux groupes, d'un côté les fonctions de routage et modules associés qui peuvent remplacer les fonctions de routage clés fournies par l'implémentation officielle du routeur NDN (NFD), et de l'autre les fonctions de support qui peuvent apporter des traitements de paquets supplémentaires sur un lien entre deux fonctions de routage. Dans la suite du chapitre nous utiliserons indifféremment les termes *microservice* et *module* qui se réfèrent au composant logiciel alors que le terme de *fonction* se réfère au service rendu.

6.3.1 Définitions communes et contraintes de conceptions

Avant de présenter les différents microservices, nous définissons les fondations qu'ils ont en commun. La conception des services suit trois lignes directrices. La première est d'être pleinement compatible avec le protocole NDN et ainsi être capable de communiquer avec l'implémentation officielle. La deuxième est de forcer la décomposition en services distincts le plus possible, pour intensifier les aspects positifs et négatifs d'une telle architecture. La troisième est que nos liens entre nos modules suivent une logique de pipeline pour le chaînage de fonctions. Plus précisément, un module ne connaît pas qui est avant ou après lui et peut ainsi être connecté à n'importe quel autre module, même si toutes les combinaisons ne sont pas cohérentes. Le manager central (qui sera décrit dans la section 6.4) est le seul à avoir une vue globale du réseau. D'autres architectures de management peuvent être envisagées pour l'orchestration des microservices (distribuée ou auto-managée [TBB+15]) mais vont au delà de notre problématique.

Pourtant, de nombreux défis restent à relever :

— Comment décomposer un routeur monolithique ?
— Comment lier les microservices ?
— Comment traiter les flux de paquets *Interest* et *Data* ?

Dans la description des différents modules, nous utilisons deux types de *Faces* : celles d'entrée et de sortie. Les *Faces* sont un terme utilisé par les développeurs de NDN qui représente une abstraction des interfaces. Cela fonctionne comme un socket classique avec pour différence que le protocole sur lequel NDN est implémenté est « caché », ce qui permet d'envoyer et recevoir des messages grâce à une classe commune qui s'adapte aux protocoles sous-jacents utilisés. les *Faces* d'entrée et de sortie font référence au module qui initie la connexion pour créer le lien. Une *Face* d'entrée écoute le trafic/connexion entrant. De plus, chaque microservice peut être orienté ou non. Un module orienté ne traite que les paquets *Interest* ou *Data* sur ses *Faces*, alors qu'un modules non orienté peut traiter les deux.

Nous ferons également référence à la cardinalité effective d'un module qui est le nombre de sources ou de destinations différentes qu'il peut distinguer lors du transfert, mais qui n'a aucun rapport avec le nombre d'*endpoints* qu'un module peut gérer simultanément. La cardinalité peut prendre deux valeurs :

— Une cardinalité effective de « 1 » signifie qu'un module doit être connecté à un seul autre. S'il doit gérer plus d'une connexion, il est toujours possible de transférer des paquets vers toutes ses sources ou destinations en inondant. Cela peut se produire parce que, pour une source, elle n'a pas suffisamment d'informations pour savoir à qui renvoyer la réponse, ou pour une destination, elle ne stocke pas d'informations pour router correctement le paquet ;
— Une cardinalité effective de « N », signifie qu'un module est capable de transférer des paquets vers la ou les bonne(s) source(s) ou destination(s), ce qui implique de maintenir certaines informations de routage.

6.3.2 Description des microservices

Fonctions de routage

Notre première décision clé a été de séparer les fonctions PIT et FIB. C'est un défi parce qu'elles sont connues pour être étroitement couplées pour implémenter le routage du protocole NDN. Toutefois, l'idée de scinder les fonctions réseau centrales d'un routeur en services distribués a déjà été introduite dans un rapport de CCNx. [19]. Nous avons choisi de séparer la PIT et la FIB parce qu'il peut être utile d'avoir une PIT distincte dans certains scénarios spécifiques. Par exemple, pour mettre à l'échelle la PIT seule lorsqu'elle limite les performances du réseau. Aussi, comme NFD, nous avons besoin d'une troisième fonction pour sélectionner le bon pipeline de traitement selon qu'il s'agit d' *Interest* ou de *Data* à cause de la séparation de la PIT et la FIB. Ainsi, les trois fonctions de routage ont le but suivant :

Name Router (NR) Cette fonction gère les demandes d'enregistrement de préfixes envoyées par les producteurs et effectue le routage en fonction du nom des paquets. Elle est similaire à la *Forwarding Information Base* (FIB) du routeur NDN. Ce module est le seul qui est « obligatoire » dans un réseau μNDN puisqu'il est le seul qui peut écouter les annonces de routes entrantes des fournisseurs de contenus et donc router efficacement les paquets des utilisateurs vers ces fournisseurs de contenus. Ce module est orienté et possède deux types d'interfaces d'entrée : les

19. https ://github.com/PARC/CCNxReports/raw/master/2014/5.1 CCNx 1.0 Implications for Router Design.pdf

Faces qui gèrent les utilisateurs (ou d'autres modules) et qui ont une cardinalité de 1 et les *Faces* qui gèrent les fournisseurs de contenus et qui ont une cardinalité de N. Les *Faces* de sortie, quand à elles, ont une cardinalité de N.

Backward Router (BR) Cette fonction garde la trace des paquets *Interest* qui la traverse afin de retransmettre plus tard les paquets *Data* aux bons utilisateurs. Puisqu'elle stocke les demandes des utilisateurs, elle peut également gérer un mécanisme de retransmission. Cette fonction est similaire à la *Pending Interest Table* (PIT) dans dans un routeur NDN. Elle est orientée de sorte que les paquets *Interest* ne peuvent arriver que par les *Faces* d'entrée et les paquets *Data* ne peuvent arriver que par les *Faces* de sortie. La cardinalité d'entrée et de sortie est respectivement N et 1.

Packet Dispatcher (PD) Cette fonction a pour but de séparer les différent trafics (utilisateurs et fournisseurs de contenus) en transmettant les paquets *Interest* et *Data* sur des *Faces* différentes. Elle est utilisée pour éviter des drops de paquets lorsque qu'elle est chaînée avec des modules orientés, qui ne traitent qu'un type de trafic sur leurs *Faces* d'entrées. NFD traite les paquets d'une manière assez proche dans le but de choisir le bon pipeline (*Interest* ou *Data*) pour le traitement des paquets. Nous pensons que cette fonction est plus adaptée à la périphérie du réseau pour gérer le trafic externe qui n'est pas conscient de la gestion spécifique du trafic effectuée par μNDN mais ne devrait pas être utile à l'intérieur du réseau. Cette fonction, n'est pas orienté et sa cardinalité est N pour chacun de ces côtés, cela est possible car le comportement d'un *Packet Dispatcher* est proche d'un proxy et ainsi, chaque tuple de *Faces* peut être vu comme une session.

Fonctions de support

Ces microservices peuvent être considérés comme des modules *on-path* parce qu'ils ne sont pas utilisés pour le routage des paquets du réseau. Ils ne sont pas orientés et ont une cardinalité de 1 pour leurs entrées et sorties. Ils peuvent améliorer les performances, la fiabilité ou la sécurité du réseau. Deux de ces fonctions sont extraites du routeur monolithique NFD :

Content Store (CS) Cette fonction a pour but de stocker les paquets *Data* récents qui la traversent afin de les réutiliser plus tard. Puisqu'elle est optionnelle dans le routeur NFD (la taille CS peut être mise à zéro), il est logique d'en faire une fonction autonome.

Strategy Forwarder (SF) Cette fonction est utilisée pour transférer des paquets vers une ou plusieurs destination(s) sélectionnée(s) en fonction d'une stratégie donnée. Dans le routeur NFD, elle est appliquée après un *match* au niveau de la FIB et la stratégie peut être définie pour n'importe quel préfixe enregistré dans la FIB et est hiérarchique. L'extraction de cette fonction permet des règles de forwarding plus générales car elle peut être utilisée entre n'importe quel module (ne dépend plus des *Faces* sélectionnées après un FIB matching). Les stratégies peuvent être simples et utilisées à bas niveau comme l'équilibrage de charge, le basculement, etc. ou d'autres plus spécifiques à NDN. La seule stratégie qui reste dans la FIB est la stratégie multicast (déjà définie dans NFD) qui achemine les paquets *Interest* vers toutes les destinations qui ont enregistré un préfixe. Les règles de stratégie ne sont appliquées que pour les *Faces* de sortie et la cardinalité dépend de la stratégie réelle en usage.

TABLE 6.1 – Résumé des caractéristiques des modules

Nom	**Fonction**	**Orienté**	**cardinalités d'E/S**
Name Router	Route les paquets *Interest*	Oui	1/N
Backward Router	Distribue les paquets *Data*	Oui	N/1
Packet Dispatcher	Sépare les deux types de trafics	Non	N/N
Content Store	Cache les paquets *Data*	Non	1/1
Strategy Forwarder	Forward les paquets NDN	Non	1/1 or N
Signature Verifier	Vérifie la signature des paquets	Non	1/1
Name Filter	Filtre les paquets par leur nom	Non	1/1

Nous pouvons également profiter du paradigme NFV pour proposer des fonctions plus spécifiques. Par exemple, des fonctions de sécurité trop exigeantes ou trop spécifiques peuvent être implémentées sous forme de microservices :

Signature Verifier (SV) cette fonction vérifie la signature des paquets *Data*. La vérification de la signature des paquets n'est pas effectuée par les routeurs NDN en raison de l'absence de modèle de confiance universel et du coût élevé de la vérification. Dans μNDN, cette fonction peut être configurée avec un modèle de confiance à suivre et placée dans un processus différent ou même dans un serveur différent ce qui réduit son impact sur les autres composants du réseau. Pour stocker les clés, cette fonction peut utiliser à la fois un stockage local comme référentiel de clés publiques ou une base d'information publique NDN si elle existe. Cette fonction rapporte ensuite le nom de chaque paquet défaillant à une entité supérieure qui peut déployer des contre-mesures si nécessaire. Si la signature des paquets ne peut pas être vérifiée à la vitesse de la ligne, le module peut effectuer une vérification statistique. Trois règles de filtrage peuvent aussi être définies selon que le paquet rate la vérification de signature, n'est pas signé avec une clé connue ou ne contient pas de signature.

Name Filter (NF) Cette fonction supprime simplement les paquets NDN en fonction de leur nom (correspondance stricte ou partielle). Elle est utilisée pour éviter des paquets spécifiques dans le réseau en filtrant le trafic à la périphérie de celui-ci, ou pour limiter l'usage d'un chemin donné pour un service en cas de multiples chemins disponibles (par exemple, pour limiter la diffusion de vidéos pour un service spécifique à un seul chemin).

Ces deux fonctions peuvent également communiquer entre elles afin de créer quelque chose de similaire à un pare-feu en faisant en sorte que le *Signature Verifier* ajoute directement des règles dans un *Name Filter* dans le but de réduire la vérification de faux paquets. En raison de leur nature « *on-path* », ces fonctions peuvent être facilement utilisées entre les nœuds de routeur NFD par exemple. Les principales propriétés de chaque microservice sont résumées dans le tableau 6.1.

6.3.3 Exemples de réseaux par composition de microservices

Dans la Figure 6.1, on peut voir une topologie μNDN qui imite les fonctionnalités réseau d'un nœud NDN monolithique qui utiliserait une stratégie *multicast*. Pour ce service, nous avons besoin des trois modules principaux (*Content Store*, *Backward Router* et *Name Router*) pour simuler les trois tables internes CS, PIT et FIB. Nous utilisons aussi un *Packet Dispatcher* pour gérer les connexions externes et la sélection de pipeline basée sur le type de paquet envoyé par

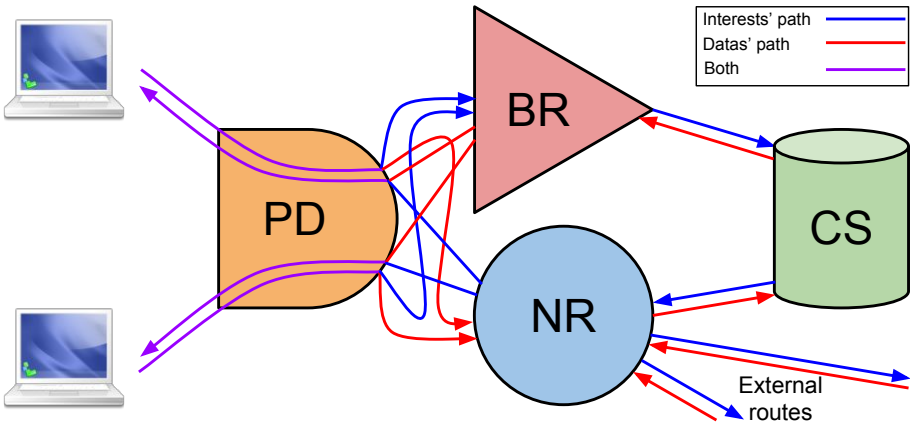


FIGURE 6.1 – Exemple de chaîne de microservices équivalente à un nœud NFD utilisant la stratégie multicast

les clients (utilisateurs et fournisseurs de contenus), ce qui permet aux clients de se connecter à notre réseau de la même manière qu'un réseau à base d'instances NFD (sans avoir à spécialiser leurs *Faces* pour un type de trafic particulier). Dans notre cas, le module *Name Router* n'est pas capable d'utiliser des stratégies de *forwarding* car nous avons réduit cette fonction en créant un module à part entière, mais cette fonction peut être rajoutée en ajoutant juste après le *Name Router* un (ou plusieurs) *Strategy Forwarder(s)*. Par défaut, la stratégie utilisée par le *Name Router* est d'envoyer les paquets *Interest* sur toutes les *Faces* qui ont enregistré au moins un préfixe du nom de ces paquets, ce qui est équivalent à la stratégie *multicast* dans NFD.

La Figure 6.2 donne un exemple d'un réseau plus complexe constitué de 5 microservices. Cette figure comporte notamment trois points d'intérêts : le premier est un cas pratique d'utilisation du chaînage que l'on a vu dans l'exemple précédent. Nous pensons que ce genre de chaînage est plus enclin à être placé en bordure de réseau pour proposer des *Faces* bidirectionnelles aux clients. Le second point d'intérêt est le fait que l'on n'est pas obligé de suivre l'ordre par « défaut » (même pipeline que dans NFD) mais d'utiliser que ce qui est nécessaire. Dans la topologie présentée dans la figure 6.2 (à droite), un client ne peut pas utiliser une même *Face* pour générer à la fois du trafic de type utilisateur (*Intetest*) et fournisseur de contenu (*Data*) mais ce client peut tout de même faire les deux rôles en même temps si celui-ci utilise au moins deux *Faces* distinctes, une pour chaque type de trafic. Le dernier point d'intérêt est le fait de vouloir relier deux *Name Routers* ensemble. Cela peut créer des boucles dans le cas où un préfixe est accessible par les deux NRs simultanément. Pour éviter que les paquets ne bouclent dans le réseau, il faut placer un *Backward Router* entre les deux *Name Routers* pour éviter cette tempête de broadcast (cette action peut être automatiquement faite par le manager).

6.4 Management et orchestration des microservices

6.4.1 Le manager

Pour pouvoir utiliser tout le potentiel d'une architecture à base de microservices, qui s'appuie fortement sur la reconfiguration dynamique du réseau en fonction des besoins opérationnels, un manager performant est nécessaire (ou *VNF manager* dans la nomenclature NFV). Nos modules ont été initialement développés sans manager. Ainsi, il reste possible de les utiliser sans manager pour des réseaux simples, mais à mesure que le réseau grandit et se complexifie, le besoin d'avoir

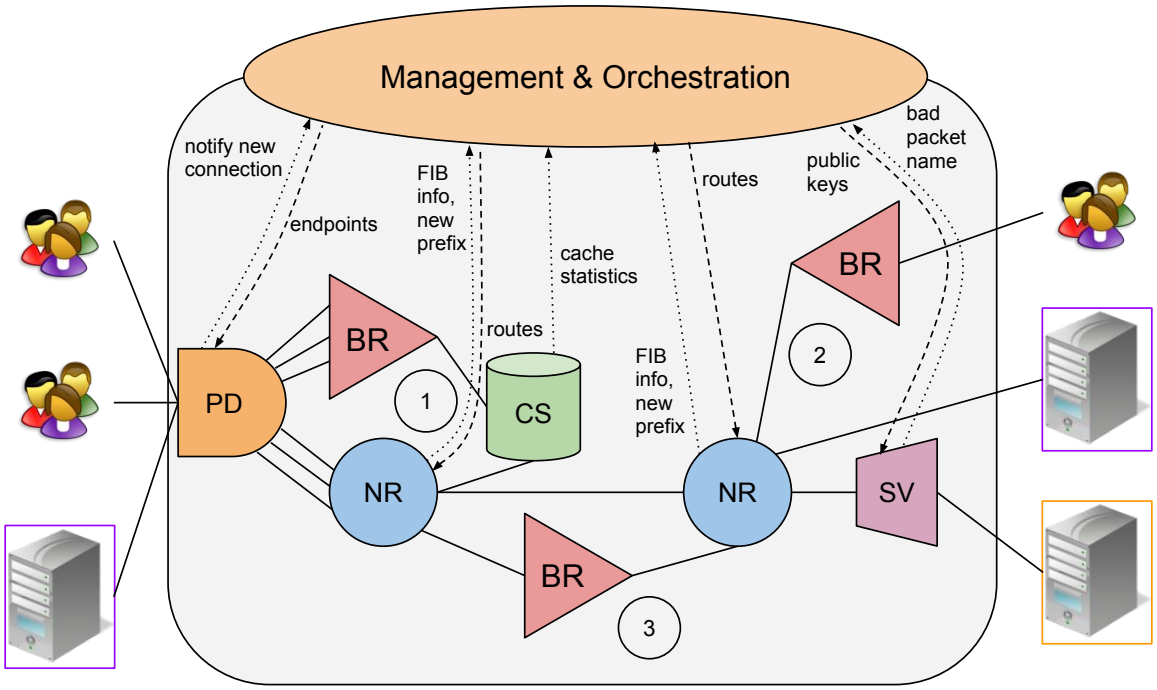


FIGURE 6.2 – Exemple d'un petit réseau μNDN managé

un ou plusieurs programmes tiers pour le gérer augmente lui aussi. Nous pensons que le manager doit être en charge de tout un système autonome (AS) qui utilise μNDN ou du moins un point de présence (PoP). Rien n'empêche la création d'une entité supérieure au manager qui permettrait de gérer et coordonner plusieurs PoP. En particulier, le manager est en charge d'opérations clés pour garantir les performances et la sécurité du réseau :

— Déployer et chaîner dynamiquement les microservices où est quand cela est nécessaire ;
— Mettre à jour la configuration des microservices en temps réel ;
— Mise à l'échelle des microservices limitant les performances du réseau ;
— Déployer des contre-mesures en cas d'attaques.

Chaque module possède une interface de management connectée au manager à travers le plan de contrôle. Cette interface peut être implémentée de différentes façons, par exemple en utilisant le nom des paquets *Interest* comme commandes, comme cela est fait avec le routeur NDN, ou encore en utilisant un protocole plus standard comme une API HTTP REST. Si nous voulons suivre le paradigme NFV, il serait plus approprié d'utiliser la seconde option car la première est spécifique aux protocoles ICN. Pour réagir aux événements, le manager a besoin de stocker et de traiter des informations remontées par les microservices. La Table 6.2 liste quelques métriques qui nous semblent utiles à surveiller. Ces valeurs sont utilisées par le manager et/ou l'opérateur du réseau pour améliorer la qualité de service (QoS) du réseau. Par exemple, les paquets *Data* non sollicités et les statistiques de cache sont utilisées pour détecter certaines attaques pour des réseaux ICN comme l'attaque par empoisonnement de contenus (CPA), alors que les statistiques des *Faces* ainsi que des routes sont utilisées pour adapter la topologie du réseau dans le temps.

À l'intérieur d'un réseau managé, un protocole de propagation de routes comme NLSR [HAA+13] n'est pas obligatoire, car le manager peut prendre lui-même en charge cette fonctionnalité de propagation des routes enregistrées par les fournisseurs de contenus. Il utilise ainsi des routines pour propager les routes aux autres *Name Routers* du réseau à la manière d'un

TABLE 6.2 – Métriques suggérées à observer pour chaque microservice

Nom	**Valeurs**
Name Router	Statistiques des routes
Backward Router	Paquets *Data* non sollicités Paquets *Interest* retransmis
Packet Dispatcher	Statistiques du trafic utilisateur
Content Store	compteurs *Hit* et *Miss*
Signature Verifier	Nom des paquets ayant ratés la vérification de signature
Name Filter	Compteur de drops de paquet

contrôleur SDN chaque fois qu'un changement est notifié par un module *Name Router* où lors de la création d'un nouveau lien impliquant (directement ou indirectement) un module *Name Router*. En effet, le manager a juste à vérifier le statut des routes connues (routes statiques) et de changer la configuration des modules *Name Router* qui seront concernés par ces changements (routes dynamiques). Plus précisément, tout module *Name router* qui peut atteindre le module qui a déclenché la procédure sera concerné par ces changements. Pour la gestion des routes avec les réseaux extérieurs, le déploiement de microservices spécifiques aux types de protocoles de routage utilisés, comme NSLR, qui seront déployés en bordure du réseau managé est une bonne solution pour obtenir une communication des routes qui soit indépendante du protocole de routage sous-jacent entre les microservices et le manager.

Pour notre architecture, nous utilisons un management centralisé qui est suffisant pour nos besoins. Mais il est tout à fait possible de d'utiliser des techniques plus avancées de management comme [TBB+15] pour éviter d'avoir un unique point de défaillance mais cela est laissé pour des travaux futurs.

6.4.2 Changements dynamiques du réseau

Une des propriétés les plus intéressantes d'un réseau managé à base de microservices est sa capacité à dynamiquement évoluer, en particulier en mettant à l'échelle les modules les plus lents en cas de congestion, ou en ajoutant des fonctions supplémentaires à la volée en fonction des besoins opérationnels. Concernant la mise à l'échelle, deux modules sont généralement plus lents que les autres : *Signature Verifier* et *Backward Router* (confirmé par notre évaluation Section 6.6). Le premier car la vérification de signatures est connue pour être coûteuse en calculs, et le second car sa nature *stateful* fait qu'il est la partie la plus lente du routeur NDN. De plus, ces deux cas sont représentatifs de la façon de procéder à la mise à l'échelle d'un module selon leur cardinalité, respectivement de 1 et N respectivement pour le module *Signature Verifier* et *Backward Router*.

La Figure 6.3 illustre la façon attendue d'une mise à l'échelle pour une fonction 1/1 (*Signature verifier* dans cette exemple). La fonction mise à l'échelle doit être entourée par un module *Strategy forwarder* dont la stratégie est définie sur *load balancing* pour distribuer les paquets entre tous les modules *Signature verifier* et un module *Backward Router* pour agréger le trafic des différentes instances de la fonction mise à l'échelle. De cette façon, l'ensemble peut être vu comme une boîte avec les mêmes relations et contenant la fonction désirée. Par définition [Fow14], les microservices peuvent aussi communiquer directement entre eux. Cette particularité peut être intéressante lors d'une mise à l'échelle d'une fonction réseau pour certains modules comme le *Content Store* : une fois le passage à l'échelle réalisé les modules sous-jacents peuvent collaborer entre eux pour éviter

d'avoir des entrées dupliquées dans la grappe locale qu'ils forment.

La Figure 6.4 illustre une méthode pour réaliser la mise à l'échelle de la fonction *Backward Router*. Dans le cas de cette fonction, la méthode précédente ne peut pas être utilisée car cela déplacerait seulement le *bottleneck* au niveau du module *Backward Router* supplémentaire utilisé pour regrouper le trafic et assurer la cohérence de la connectivité. Un module supplémentaire ayant les mêmes propriétés mais en effectuant un traitement plus léger, en utilisant par exemple des filtres de bloom, serait nécessaire pour rester en accord avec les contraintes relationnelles définies. Ainsi, il est toujours possible de réaliser la mise à l'échelle de cette fonction avec les modules que nous avons développés mais cela forcera le prochain module à dupliquer le trafic de retour vers les différentes instances de la fonction, ce qui peut impacter les performances de ce module. Du côté des entrées, des modules *Strategy Forwarder* seront déployés, un par source entrante, et dont la stratégie est définie sur *load balancing* pour respecter la cardinalité de la fonction mise à l'échelle. Avec cette méthode, il est possible d'obtenir du trafic dupliqué en cas de retransmission mais cela peut se résoudre en ajoutant un filtre au niveau des modules *Strategy Forwarder* pour ne pas transmettre les paquets dupliqués.

Pour les fonctions qui stockent des informations dynamiquement comme les fonction *Content Store* et *Backward Router*, une synchronisation entre les modules d'une même fonction mise à l'échelle peut être nécessaire, surtout lorsqu'une réduction d'échelle se produit. Dans ce cas, le manager ordonne aux modules *Strategy Forwarder* de ne plus transmettre les paquets *Interest* au module qui va être retiré et lui demande de transmettre sa table aux autres modules avant de le retirer pour limiter la quantité d'informations susceptible d'être perdue et pouvant causer des retransmissions dans le réseau. Il est aussi possible pour la grappe d'une fonction mise à l'échelle d'utiliser une mémoire partagée, avec des solutions comme etcd [20].

Il existe trois stratégies (séquence d'actions) possibles pour insérer un nouveau module dans une chaîne existante. Pour les expliquer, nous considérons le cas où deux modules M1 et M2 sont déjà reliés entre eux et où un troisième module M3 est inséré entre les deux précédents. Dans les deux premières stratégies, le lien entre M1 et M2 est supprimé en premier, il faut donc choisir entre jeter les paquets entrants ou les stocker en attendant la mise en place du nouveau chaînage. Enfin, de nouveaux liens sont mis en place entre M3 et M2 et entre M1 et M3 pour obtenir la nouvelle chaîne. La troisième stratégie consiste à reporter la suppression du lien M1-M2 pour créer directement les liens M3-M2 et M1-M3, et seulement supprimer le lien M1-M2 à la fin de l'opération. Cette séquence peut dupliquer le trafic lors de la dernière étape, a l'avantage de ne jamais casser la chaîne pendant le processus d'insertion.

6.5 Implémentation de notre architecture

Concernant les considérations purement techniques, les microservices sont écrits en C++ et le manager en Python [21], le tout étant disponible en open-source [22]. Pour la partie web du manager, utilisant l'API REST, la visualisation de la topologie est réalisée avec l'aide du script D3.js et la liste des actions AJAX, sous forme de formulaires, est réalisée avec JQuery.

6.5.1 Implémentation des microservices

Les microservices fonctionnent actuellement en *overlay* d'IP parce que cela permet un prototypage plus rapide que s'ils avaient été développés directement sur Ethernet. Nos modules

20. https ://coreos.com/etcd/
21. avec l'aide des libraires Networkx, Twisted et Klein
22. https ://github.com/Kanemochi/NDN-microservices

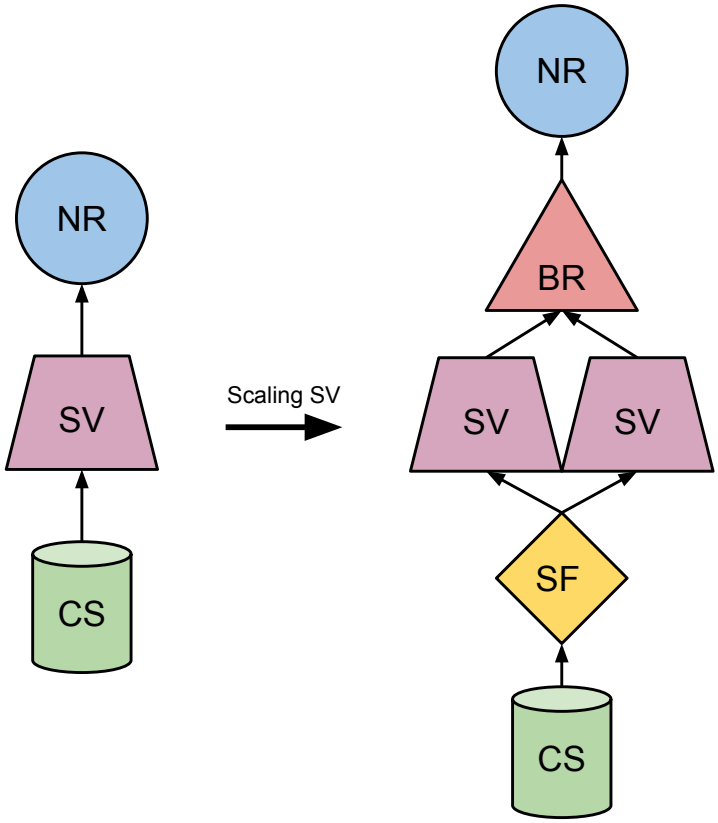


FIGURE 6.3 – Procédure de mise à l'échelle d'un module *Signature Verifier* lorsque l'on suit les contraintes de relation

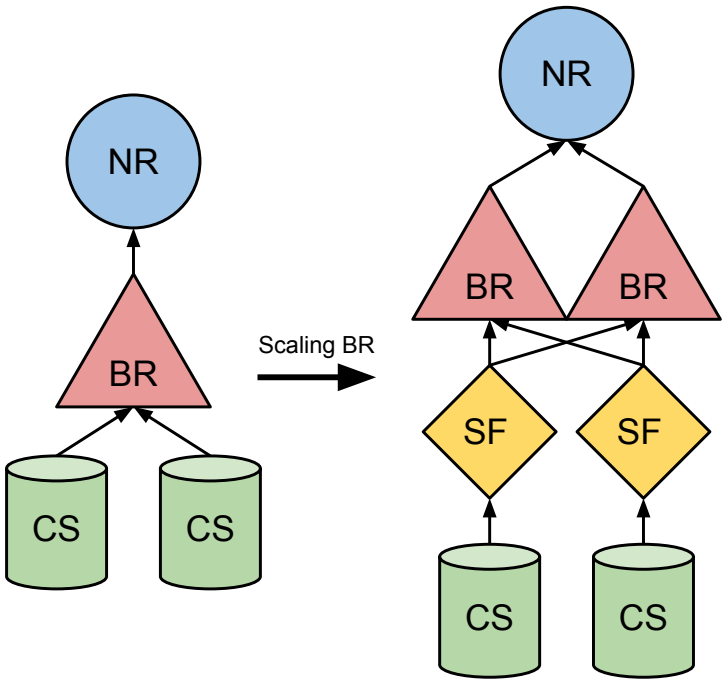


FIGURE 6.4 – Possible procédure de mise à l'échelle d'un module *Backward Router*

peuvent actuellement communiquer indifféremment sur TCP ou UDP pour transmettre des paquets NDN, mais nous supporterons les communications sur Ethernet dans un futur proche. Tous les microservices sont actuellement monothread mais certains d'entre eux pourraient être facilement parallélisés, même si la mise à l'échelle les modules les plus lents via le manager permet déjà d'allouer plus de ressources à une fonction. En pratique, les modules sont libres de se connecter à n'importe quel nombre d'*endpoints* et ne suivent donc pas strictement la cardinalité définie dans la section 6.3, qui est uniquement appliquée par le manager. Cela peut conduire à d'autres modèles intéressants de chaînes de fonctions à l'avenir, même si nous n'en avons pas exploré toute l'étendue.

La plus longue correspondance de préfixe utilisée par les fonctions BR, NR et NF et la plus courte correspondance de suffixe utilisée par la fonction CS sont implémentées grâce à une structure de données arborescente N-aire indexée utilisant le nom NDN des paquets comme clé. L'index est d'abord vérifié pour essayer de gagner du temps lorsqu'une correspondance stricte se produit. Les autres modules utilisent des structures de données plates comme des tableaux ou des hashmaps.

Le comportement de la fonction CS est légèrement simplifié par rapport au routeur NDN. Nous avons apporté les modifications mineures suivantes qui empêchent seulement les utilisateurs de récupérer des paquets périmés :

— Concernant les paquets *Interest*, nous consultons le champ *MustBeFresh* pour décider si le cache doit être utilisé : si le champ est défini à « *true* », le cache ne sera pas utilisé ;
— Concernant les paquets *Data*, nous consultons le champ *freshness* avant de prendre la décision de mettre en cache le paquet : si le champ est défini à zéro (ou n'est pas présent), le paquet ne sera pas mis en cache.

6.5.2 Implémentation du manager

Notre manager peut être divisé en cinq parties : (1) le modèle, représentation du réseau en tant que graphe, (2) le moteur d'orchestration, (3) l'interface de management des modules, (4) la gestion des clés publiques et (5) l'API REST pour interagir avec le manager.

1. La représentation de la topologie du réseau prend la forme d'un graphe orienté qui stocke toutes les informations requises pour le management, par exemple : les routes des *Name routers*, des informations comme l'usage CPU de chaque module, etc. Des procédures sont aussi associées à cette partie comme par exemple la propagation des routes.

2. Le moteur d'orchestration assure que les images des microservices, qui sont exécutées dans des conteneurs individuels, ainsi que les réseaux virtuels qui seront utilisés sont bien déployés. Pour ce faire, le moteur va demander à l'infrastructure de réseau virtuel (ici, Docker Engine) la création ou la suppression des conteneurs demandés. Cette partie est aussi en charge de récupérer à intervalles réguliers les statistiques des conteneurs pour la prise de certaines décisions comme la mise à l'échelle de modules.

3. L'interface de management des modules est utilisée pour véhiculer les informations entre le manager et les microservices. Elle identifie les conteneurs (association nom/IP) et envoie des messages de management asynchrones ;

4. La gestion des clés publiques est en charge de la synchronisation des clés avec les différentes instances de *Signature Verifiers*. Elle a aussi la charge de la vérification de signature lorsqu'une demande de route est demandée par un fournisseur de contenus avant de mettre à jour les tables de routage.

5. L'API REST permet à un logiciel externe de récupérer les informations contenues dans le graphe du manager mais propose aussi sous forme de service web une représentation dynamique du réseau et une interface permettant d'exécuter les actions proposées par l'API.

6.5.3 Communication avec les modules

Pour la communication entre les différentes entités, les messages sont émis sous forme de messages JSON et sont envoyés à travers une communication UDP qui a vocation à évoluer en véritable API HTTP REST pour plus de fiabilité.

Pour la partie manager, tous les messages contiennent au moins un champ « action » et « id ». Le champ « action » est utilisé par les modules pour mapper des mots-clés à des fonctions. Le champ « id » est un entier qui n'est utilisé que par le manager, un module va juste copier cette valeur dans sa réponse et le manager l'utilisera pour savoir à quelle requête cette réponse est associée. Selon le type des modules, plusieurs types de messages sont possibles. Tous les modules acceptent des messages *add/del_face* qui permettent de donner l'ordre de créer une nouvelle *Face* ou de supprimer une existante. En cas de création, l'ID de cette Face est retourné en plus du statut de réussite pour que le manager puisse mettre à jour son graphe avec les bonnes informations (cela peut aussi déclencher d'autres actions si besoin, comme la propagation de routes). Tous les modules acceptent aussi les messages permettant d'éditer leur configuration (certaines valeurs restent exclusives à certains modules) ou de récupérer leur configuration actuelle. Pour tout changement de configuration, le module va retourner la liste des champs dont la valeur a changé. Dans le cas des *Name Routers*, ils peuvent en plus accepter des messages *add/del_routes* qui permettent de changer la configuration des routes pour une *Face* donnée. De la même façon, les modules *Signature Verifiers* acceptent en plus des messages *add/del_keys* qui sont propres à la gestion des clés publiques. Les listings 6.1 et 6.2 donnent un exemple d'échange de messages dans le cas d'une requête *add_face* envoyée par le manager.

Listing 6.1 – Exemple d'un message « add_face »

```
{
    "action": "add_face",
    "id": 1,
    "layer": "tcp",
    "address": "172.18.0.2",
    "port": 6363
}
```

Listing 6.2 – Exemple d'une réponse pour le message « add_face »

```
{
    "name": "NR1",
    "type": "reply",
    "action": "add_face",
    "id": 1,
    "status": "success",
    "face_id": 1
}
```

Les modules peuvent envoyer des messages similaires de leur propre initiative avec le type « request » ou « report » (certains modules supportent les deux stratégies de monitoring : passive et active). Dans la situation actuelle, nous avons défini deux types de message de type « request », le premier est « route_registration » et est utilisé lorsqu'un paquet *Interest* servant à enregistrer un préfixe est reçu par un *Name Router*. Ce message contient, en plus des champs traditionnels illustrés dans le listing 6.2, le champ *KeyLocator* du paquet *Interest*, le type de signature et sa valeur, et la partie du message signée (tous encodés en base64). Ces champs seront utilisés par le manager pour vérifier que la signature est valide et retournera sa décision au module concerné. Le second message de type « request » est « new_session » qui est envoyé lorsque qu'un nouveau client se connecte à un *Packet dispatcher*. Le message contient l'*endpoint* (selon le protocole sous-jacent : IP/port, adresse MAC, etc...) du client pour que le manager décide s'il est accepté

ou non et où forwarder son trafic.

Pour demander le rapport périodique des métriques supervisées, le manager doit d'abord spécifier au module concerné où il doit envoyer ses messages et le délai minimal entre chaque message. Parmi les valeurs remontées, le *Content Store* rapporte deux compteurs (*hit* et *miss*), le *Signature Verifier* rapporte les noms des paquets ayant échoués la vérification de signature. Quelques modules ne tiennent pas compte du délai entre message car ceux-ci sont considérés comme critiques et doivent être traités immédiatement. C'est le cas avec les *Name Router* qui peuvent transmettre un message « producer_disconnection » pour notifier qu'un *producer* s'est déconnecté et ainsi mettre à jour les routes dans le réseau en tenant compte de cet événement. Il existe un message similaire pour le *Packet dispatcher* pour notifier la déconnexion d'un client.

Listing 6.3 – Exemple de message pour notifier le statut du cache d'un *Content Store*

```
{
    "name": "CS1",
    "type": "report",
    "action": "cache_status",
    "hit_count": 125000,
    "miss_count": 100000
}
```

6.5.4 Management des routes

Puisque que nous avons un réseau managé, nous n'avons pas de réel besoin de protocole de routage interne comme NLSR [HAA+13] car le manager gère de manière centralisée, à la manière d'un contrôleur SDN, les routes disponibles sur le réseau et envoie les changements directement aux modules concernés. Cette action est réalisée chaque fois qu'un fournisseur de contenus envoie une demande de route ou quitte le réseau, ou encore, que des modules sont reliés entre eux. Le manager vérifie le statut de toutes les routes connues (routes statiques) et change la configuration des *Name Routers* qui sont concernés par ces changements (routes dynamiques). Plus précisément, la gestion des routes est réalisée par trois procédures :

appendExistingRoutes Lorsque qu'un module se connecte à un autre (source), le manager va regarder quelles sont les routes qui peuvent être atteintes par le module auquel le premier s'est connecté. Les routes ainsi récoltées ne seront poussées que si le module source est de type *Name Router*. Après cette étape, peu importe le type du module source, le manager va ensuite appliquer la procédure *propagateNewRoutes* comme si le module source était le propriétaire des routes récoltées pour pouvoir récursivement appliquer les changements aux possibles modules derrières celui-ci.

propagateNewRoutes Cette routine est généralement appelée lorsqu'un fournisseur de contenus a réussi à enregistrer un préfixe. Elle va regarder pour tous les autres *Name Routers* si ceux-ci sont capables d'atteindre le *Name Router* qui est concerné par la création de la nouvelle route, puis, pour chaque chemin possible, le manager va comparer la liste des routes existantes et appliquera le changement si nécessaire.

propagateDelRoutes Cette routine est appelée lorsque qu'une route est retirée d'un *Name Router* ou que celui-ci est retiré du réseau. Pour tous les *Name Routers* qui ont au moins un chemin vers ce *Name Router* et qui ont au moins une route dynamique servie par le *Name Router*

TABLE 6.3 – Débit maximum moyen pour chaque microservice

Module	Throughput (Mbps)		
	Bare-Metal	Container	Difference
Name Router	2,144	1,935	-10.5%
Backward Router	1,480	1,380	-6.8%
Packet Dispatcher	2,334	2,081	-10.8%
Content Store (freshness = 0)	2,224	1,997	-10.2%
Content Store (freshness > 0)	1,151	1,010	-12,3%
Content Store (from cache)	3,431	2,852	-16,9%
Strategy Forwarder	2,281	2,058	-9.8%
Signature Verifier (RSA2048)	665	630	-5.3%
Signature Verifier (ECDSA256)	121	118	-2.5%
Name Filter	2,184	1,971	-9.8%

en cause, le manager va regarder si les préfixes venant de ce chemin sont toujours accessibles, sinon les routes devenues inaccessibles seront notifiées aux *Name Routers* concernés.

6.6 Évaluation de notre solution

6.6.1 Environnement expérimental

Pour les expériences si dessous, nous utilisons un serveur DELL PowerEdge R730 articulé autour de deux processeurs Intel Xeon E5-2630v3 (2x8 cœurs) avec les technologies Hyper-Threading et Turbo Boost activées. Le serveur possède aussi 128GB de mémoire vive et deux SSD de 400GB chacun montés en RAID0 pour les besoins du système d'exploitation et des conteneurs Docker. Nous utilisons la version 18.03 de Docker CE ainsi que la version 0.6.1 de la librairie ndn-cxx.

6.6.2 Tests unitaires de nos microservices

Dans le but de connaître la quantité de ressources CPU utilisée par chaque module, mais aussi pour identifier lesquels ont des risques de devenir des goulots d'étranglement, nous avons évalué le débit de chacun des types de module que nous avons développés. Durant cette expérience, nous utilisons une topologie très simple et qui peut être considérée comme étant le meilleur cas d'usage : les modules sont placés entre un *producer* et un *consumer* qui utilisent notre programme NDNperf [MCF16b]. Nous avons aussi défini l'affinité des programmes pour qu'ils soient exécutés sur des cœurs CPU précis pour optimiser l'exécution des threads. Il faut noter que les modules ne sont pas exécutés dans un conteneur pour cette expérience, ces différences font que le débit reporté est un peu plus important que celui possible dans un environnement virtualisé. La Table 6.3 nous donne les débits moyens atteints par chacun des modules, certains sont donnés sous différentes configurations lorsque cela est pertinent.

En se basant sur ces résultats, nous pouvons voir que les modules *Backward Router* et *Content Store* (dans un cas particulier) sont les plus enclins à devenir bottleneck du réseau. Pour le cas spécifique du *Content Store*, sa capacité à devenir un bottleneck est très liée au ratio de paquets qui sont réutilisés pour alimenter le réseau. Comme les différentes opérations possibles dans un *Content Store* ont des coûts différents, le plus élevé étant quand le *Content Store* a à mettre

TABLE 6.4 – Comparaison des performances entre NFD et son équivalent à base de microservices

	Microservices				NFD
	PD	CS	BR	NR	
%CPU core usage	100	59	89	64	100
Throughput (in Mbps)	776				527
Latency (in ms)	2,63				3,88

en cache tous les paquets *Data* qui le traversent sans avoir l'occasion de les réutiliser. Ainsi le cache peut devenir bottleneck à la place d'un *Backward Router* si le ratio de cache hit n'est pas suffisamment important, selon les chiffres rapportés dans la Table 6.3, nous avons identifié dans notre cas qu'il faut au moins un ratio de 20% pour que le *Content Store* ne deviennent pas bottleneck du réseau dans un cas où tous les paquets Data traversant le *Content Store* sont mis en cache. Comme déjà suspecté, la vérification de signatures est une tâche lourde mais celle-ci n'est pas critique car cette vérification est plus encline à être placée en bordure de réseau et à la demande. Ainsi, cela ne limitera que le trafic entrant dans le réseau qui a été détecté suspicieux ou qui n'est pas de confiance.

Dans une autre expérience présentée dans la Table 6.4, nous pouvons voir une comparaison entre la topologie présentée dans la Figure 6.1 et une instance de NFD. Il n'est pas très juste de comparer directement les deux architectures car les fonctionnalités qu'elles proposent ne sont pas identiques, mais cela nous permet d'avoir une première idée de ce dont est capable notre architecture. Dans cette expérience, tous les modules n'ont qu'un seul thread et nous pouvons voir que notre chaîne μNDN est plus rapide que l'instance NFD équivalente en terme de débit et latence (environ 50% plus rapide). Cependant, cela est possible au prix d'une utilisation CPU bien plus importante (3 fois plus de cycles utilisés). Dans cette configuration particulière, le module *Packet Dispatcher* est le bottleneck de notre expérience car celui-ci doit gérer deux fois le trafic généré par le consommateur et le fournisseur de contenus alors que les autres modules ne le font qu'une seule fois car le consommateur et le fournisseur de contenu son du même « côté » du réseau. Néanmoins le module *Packet Dispatcher* n'est pas un module complexe et pourra être facilement multi-threadé. Si nous prenons le cas où le *Packet Dispatcher* n'est pas bottleneck, par exemple lorsque le fournisseur de contenus est directement connecté au *Name Router*, nous avons mesuré un débit de 968 Mbps avec des coûts CPU de 67%, 100% et 71% pour respectivement le *Content Store*, *Backward Router* et *Name Router*.

6.6.3 Mise à l'échelle dynamique

Pour expérimenter les propriétés de mise à l'échelle de μNDN lorsque des pics de trafic surviennent, nous avons construit un réseau avec 3 modules dans l'ordre suivant : un *Content Store*, un *Backward Router* et un *Name Router* (comme dans la Figure 6.4 mais avec un seul CS). Dans cette expérience, le *Content Store* est utilisé uniquement en tant que simple proxy car aucun paquet ne sera mis en cache. Nous avons besoin d'une séquence d'au moins 3 modules pour qu'une mise à l'échelle puisse se produire car nous avons défini une règle pour que les modules en bordure du réseau ne puissent pas être mis à l'échelle. Nous avons également limité l'utilisation CPU des modules *Backward Router* à 67% d'un cœur CPU pour accentuer artificiellement le bottleneck et mieux voir l'effet du processus de mise à l'échelle. Chaque fois qu'une fonction remplit les conditions suivantes, celle-ci peut être mise à l'échelle (scale up) : (1) elle ne doit pas être en périphérie du réseau (pour éviter de rompre les connexions TCP en dehors du réseau géré), (2) elle doit être définie comme « scalable » (les fonctions déployées automatiquement ne sont

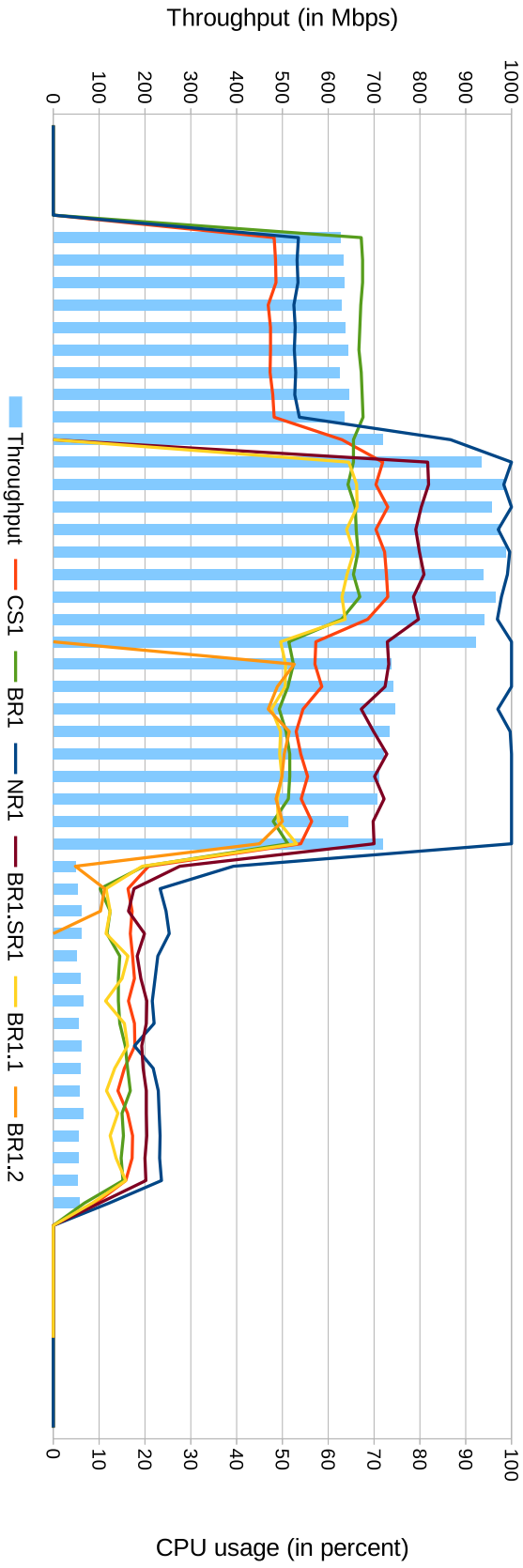


FIGURE 6.5 – Débit et utilisation CPU du réseau durant un événement de scaling

pas définies comme « scalable » par défaut), et (3) elle doit avoir une utilisation CPU dépassant 80% de la capacité totale allouée sur la dernière période de temps. Si toutes les conditions sont réunies, une routine d'orchestration sera déclenchée pour mettre à l'échelle cette fonction. De la même manière, lorsqu'une fonction mise à l'échelle a une utilisation de CPU inférieure à 20% de sa capacité, une routine d'orchestration va réduire le facteur de mise à l'échelle (scale down). Cette routine de mise à l'échelle est exécutée toutes les 20 secondes.

La Figure 6.5 montre le débit mesuré par le consommateur et l'utilisation CPU de tous les microservices qui constituent le réseau. Au début, un consommateur est démarré avec une fenêtre de réception de 32 paquets *Interests* pour surcharger le réseau, résultant en un débit moyen de 635 Mbps, limité par le *Backward Router* (BR1, qui a atteint sa capacité CPU de 67%). Ensuite, le processus de mise à l'échelle est déclenché et suit la procédure décrite dans la Section 6.4 : le manager crée d'abord un *Strategy Router* et l'insère entre le *Content Store* et le *Backward Router*. Ensuite, il déploie jusqu'à 2 modules *Backward Router* supplémentaires (BR1.1 et BR1.2) et les relie au module *Name Router* et au module *Strategy Forwarder* nouvellement déployé. Pendant le processus de mise à l'échelle, on peut remarquer que le débit augmente jusqu'à environ 960 Mbps pour un facteur d'échelle de 2 (lorsque BR1.1 est créé), mais le débit diminue ensuite à environ 715 Mbps lorsque le troisième module *Backward Router* est créé. Cela peut s'expliquer par le fait que le module *Name router* utilise déjà 100% de son cœur CPU alloué et ne peut plus utiliser de ressources supplémentaire pour transférer le trafic supplémentaire, ce qui va faire baisser le débit utile du module. Cependant, notre procédure de mise à l'échelle pourrait être améliorée en tenant compte de la topologie et de la charge dynamique des autres modules. De cette façon, il n'aurait pas ajouté le troisième module BR inutile sachant que NR était déjà à sa limite. Après cette période de stress de notre topologie, nous ralentissons le consommateur en définissant sa fenêtre de réception à 1 pour déclencher le processus de réduction du facteur d'échelle. Ceci est illustré dans le dernier tiers de la Figure 6.5 : les modules clones sont supprimés lorsque leur utilisation processeur atteint 0% (et leurs courbes respectives ne continuent pas après ce point). A la fin de l'expérience, le réseau revient dans son état initial.

6.6.4 Management de la sécurité

Dans cette dernière expérience, nous voulons tester la capacité de notre réseau à pouvoir dynamiquement déployer une contre-mesure contre une attaque par empoisonnement de cache (CPA). Nous avons construit un réseau qui laisse de la place pour une CPA, similaire à celle qui affecte NFD, comme décrit sur la Figure 6.6. Notre réseau est fait d'une chaîne composée de deux *Content Stores* en bordure du réseau et d'un *Name Router* en son centre. La faille de ce réseau est qu'un *Content Store* est directement connecté du côté du fournisseur de contenus du *Name Router*. Ainsi, lorsque qu'un bon fournisseur de contenus se connecte au réseau via ce *Content Store* et enregistre un préfixe sur le réseau, le *Name Router* va bien transmettre les paquets vers celui-ci mais via le *Content Store*, qui lui n'est pas capable de router les paquets NDN (il fait juste un broadcast). Partant de ce principe, un attaquant peut tenir compte de cette particularité et se connecter sur le même *Content Store* que le fournisseur légitime, seule façon de pouvoir traverser le *Name Router* car celui-ci ne transmet pas les paquets Data qui viennent de *Faces* qui n'ont pas enregistré de préfixe valide au nom des paquets. Comme pour le chapitre 5, l'attaquant collabore avec un client malveillant pour réaliser la CPA : ce dernier va demander les mauvais contenus qui doivent être insérés dans le cache des *Content Stores*. Le client légitime est configuré pour ne demander qu'un nombre limité de contenus différents pour obtenir un cache hit de 100% en l'absence d'attaque.

Sur la Figure 6.7, nous pouvons voir un histogramme qui représente le cache hit du *Content*

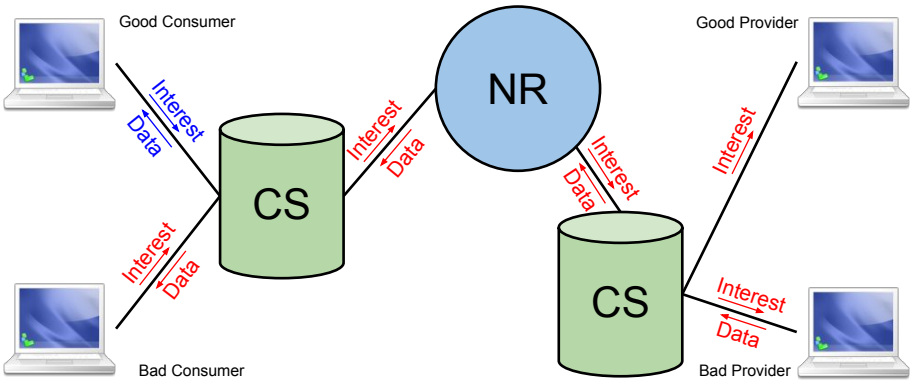


FIGURE 6.6 – Scénario pour une attaque par empoisonnement de cache

Store du côté du client et quelques courbes qui représentent l'utilisation CPU des modules (on ignore le *Content Store* côté fournisseur pour une meilleure lisibilité). Une fois que tous les contenus ont été demandés par le bon client au moins une fois (la fraîcheur des paquets Data peut être considérée comme infinie dans notre expérience), le ratio de cache-hit atteint 100%. Quand le *Content Store* est capable de répondre à toutes les demandes du client, il n'y a plus de trafic qui le traverse et cela a pour résultat de réduire l'utilisation du CPU des autres modules à 0%. Puis, nous lançons le couple d'attaquants pour réaliser la CPA. On observe après leur lancement une petite diminution du cache hit du *Content Store* mais celle-ci n'est pas suffisante dans un premier temps pour définir le trafic comme suspect et déclencher la procédure de vérification des paquets (100% à 94%). Mais sur la période suivante, on peut voir le cache hit diminuer de 94% à 30%. Ce changement est assez important pour déclencher la procédure de vérification. À ce moment-là, le manager va demander à l'orchestrateur de créer un module *Signature Verifier* entre le *Content Store* côté fournisseur et le *Name Router*. Le *Signature Verifier* est configuré avec les clés publiques connues par le manager et a pour tâche de jeter tout paquet *Data* qui échouera la vérification de signature. Son exécution rétablit rapidement le cache hit à sa valeur d'origine et l'on peut conclure que la présence du *Signature Verifier* permet de contrer l'attaque. À la fin de l'expérience, le client malveillant va arrêter son attaque à cause d'un trop grand nombre de réponses hors délais. Finalement, le manager va retirer le *Signature Verifier* et ainsi restaurer le réseau dans son état d'origine dès lors qu'aucun paquet n'échoue la vérification pendant 10 périodes de reporting (une période est définie à 2 secondes).

6.7 Discussion

Avec du recul, notre choix de conception le plus discutable est la séparation des fonctions PIT et FIB en deux modules distincts, car elle entraîne des inconvénients importants. Nous avons dû définir des *Faces* spécialisées par type de trafic entrant (*Interest* ou *Data*) qui rend ces modules « orientés » (voir Tableau 6.1) afin d'éviter la diffusion de trafic non géré (*Interest* pour la PIT et *Data* pour la FIB). De plus, la cardinalité asymétrique du module *Name Router* (1/N) et du module *Backward Router* (N/1) les rend plus difficiles à mettre à l'échelle séparément. En effet, la mise à l'échelle des fonctions de routage comme *Backward Router* (*Name Router* n'a pas été introduite mais elle peut être vue comme une fusion des deux méthodes de mise à l'échelle décrites avec la contrainte supplémentaire d'avoir des routes cohérentes entre les instances mises à l'échelle). Cela augmente donc significativement la complexité de la gestion du réseau pour un

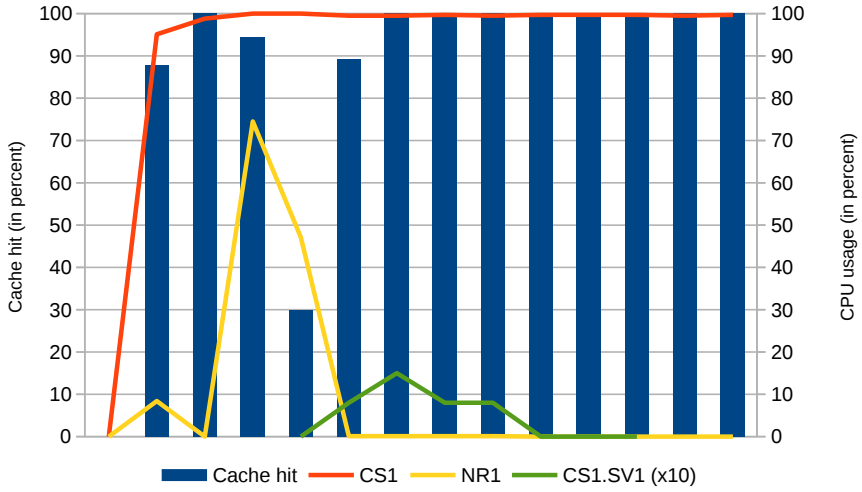


FIGURE 6.7 – Valeur du cache hit dans le temps durant une attaque par empoisonnement de cache

gain assez limité d'après notre expérience, même si notre implémentation reste à optimiser.

Ainsi, un module unifié regroupant les fonctions PIT et FIB semble être le choix le plus raisonnable pour le moment. Ce module unifié pourra effectuer les deux actions sur toutes ses *Faces*, réduisant ainsi la complexité du réseau. Une solution intermédiaire au développement d'un nouveau composant PIT+FIB consiste à regrouper nos trois fonctions de routage de base (*Name Router*, *Backward Router* et *Packet Dispatcher*) dans une seule VM ou conteneur, car il est possible dans le cadre de NFV de construire une fonction réseau virtualisée à partir de plusieurs autres. Le module *Backward Router* peut être conservé pour des scénarios très spécifiques comme dans [STPP15] où la PIT est enrichie avec des fonctionnalités de routage supplémentaires, avec par exemple du *off-path routing*) pour améliorer la diffusion des contenus. Par contre, toutes les fonctions de support « *on-path* » (y compris le module *Content Store*) peuvent facilement tirer parti des architecture à base de microservices.

6.8 Conclusion

Dans ce chapitre, nous avons présenté μNDN, une façon alternative de réaliser les fonctions de routage et de support pour NDN en utilisant une architecture à base de microservices, afin de faire profiter pleinement NDN des propriétés de NFV. Notre premier travail a été de diviser les trois fonctions clés du routeur NDN en fonctions réseau virtualisées dédiées. Pour avoir une architecture complète, nous avons également développé un manager ainsi que d'autres microservices pour couvrir les besoins importants d'un opérateur réseau en matière de performance et de sécurité. Les microservices permettent de concevoir des réseaux NDN plus efficaces en ne déployant une fonction précise qu'en cas de besoin, mais au prix d'une plus grande complexité de gestion, c'est-à-dire plus de tâches de chaînage et d'orchestration, pour construire et exploiter ce type de réseau.

Même si une comparaison directe avec le routeur de référence, NFD, est biaisée de par les caractéristiques spécifiques des deux implémentations, notre évaluation met tout de même en évidence les avantages de l'architecture à base de microservices. En effet, le fractionnement des fonctions a permis d'obtenir un débit plus élevé que l'implémentation actuelle du routeur NDN

sans trop d'efforts d'optimisation. Nous avons également montré que notre orchestrateur peut mettre à l'échelle un composant qui limiterait les performances du réseau et ajouter des modules de sécurité à la volée pour atténuer les problèmes de sécurité. Enfin, le code source des différents programmes composant μNDN est publié en open-source.

A court terme, nous allons faire en sorte que nos microservices communiquent directement par Ethernet pour éviter l'utilisation du protocole IP sous-jacent et continuer à optimiser notre architecture pour améliorer le chaînage et la livraison des paquets. Nous rendrons également notre architecture compatible avec les normes NFV comme le langage de description de services TOSCA. A plus long terme, nous explorerons davantage les possibilités offertes par μNDN en ajoutant de nouvelles fonctionnalités à l'ensemble des microservices.

Conclusion générale

Les architectures ICN ont été pensées pour répondre de manière plus efficace aux usages que nous faisons aujourd'hui d'Internet, notamment la diffusion de masse de contenus, la mobilité, etc. Cependant, leur mode opératoire est fondamentalement différent d'IP, le protocole central de nos communications informatiques, et l'adoption de telles architectures par les fournisseurs d'accès à Internet semble difficile. Nous avons présenté plusieurs architectures ICN populaires dont NDN, l'architecture ICN sur laquelle nos travaux ont été réalisés. Outre cette aspect, nous avons identifiés quatre verrous opérationnels qui nous semblent indispensables pour faciliter l'adoption des architectures ICN par les FAI : performance, sécurité, interopérabilité et déploiement. Dans le cadre de cette thèse, nous cherchons à résoudre ces quatre verrous en proposant des solutions et/ou améliorations pour chacun d'entre eux.

A ce titre, nous avons réalisé quatre contributions majeures, une pour chacun des verrous cités plus haut. Nous avons tout d'abord proposé des idées permettant de rendre plus efficace la génération de paquets NDN par un fournisseur de contenus après avoir évalué les performances des différents éléments du réseau. Nous avons ensuite réalisé une étude approfondie sur l'impact de différents types d'attaques d'empoisonnement de contenus qui permet de faire ressortir des métriques sous-jacentes à la mise en place d'un mécanisme de détection et proposé une correction pour prévenir le principal vecteur d'attaque. Nous avons conçu et implémenté des *gateways* permettant de faire le lien entre le web actuel du monde IP et le monde NDN pour pouvoir amener du trafic réel sur ce nouveau type d'architecture. Pour finir, nous proposons une implémentation d'un routeur NDN sous forme de microservices ainsi qu'un manager pour permettre un déploiement plus rapide et efficace d'un réseau NDN.

1 Résumé des contributions

Après avoir introduit les architectures ICN ainsi que leurs défis, puis présenté quatres des architectures les plus populaires dont notamment NDN, nous avons dressé un état de l'art relatifs aux différents verrous identifiés. Pour le premier d'entre eux, il apparaît que la recherche s'intéresse majoritairement à l'efficacité du routage, du cache et à la congestion des ICN mais accorde que peu d'attention aux performances applicatives offertes par le réseau. Ainsi nous évaluons dans un premier temps les performances et les usages en ressources selon différents contextes, notamment processeur, pour identifier les principaux éléments limitant. Il en ressort que le routeur peut-être un élément limitant dans certain cas, par exemple lorsque le cache n'est pas utilisé, et qu'un routeur NDN n'est pas encore capable de saturer un lien Gigabit Ethernet. Le fournisseur de contenus est un autre élément limitant lorsque la signature des paquets est nécessaire. La majorité des applications proposées étant single-thread, les débits atteignables ne nous semblent pas suffisants pour être utilisables dans des cas comme des services temps réel. Ainsi grâce à un logiciel de notre conception, NDNperf, nous proposons une version multi-threads

pour utiliser les capacités du réseau malgré le coût de la signature. Cette version montre qu'il faut 14 cœurs physiques pour saturer un lien gigabit. Au vu de ce coût élevé, nous proposons plusieurs améliorations basées : sur la signature utilisée, l'amélioration de la fonction de signature et la réduction du nombre de signatures nécessaires pour le transfert d'un contenu. Ces différentes améliorations permettent au fournisseur de contenus d'être 6,4 fois plus efficace qu'avec les paramètres par défaut.

Nous nous sommes ensuite intéressé à caractériser l'attaque par empoisonnement de contenus. La recherche s'étant majoritairement penchée sur le problème de l'attaque par inondation de paquets *Interest*, qui est l'attaque jugée la plus dangereuse, moins d'attention a été portée sur la CPA qui est pourtant également référencée comme critique. Quelques contre-mesures ont été proposées mais celles-ci restent trop spécifiques à leur exécution de la CPA et ne permettent pas de réellement caractériser cette attaque. Ainsi, nous avons étudié l'impact de différents types d'attaque par empoisonnement de contenus sur les différents éléments du réseau : une coopération entre un mauvais utilisateur et un mauvais fournisseur de contenu pour les deux stratégies *best-route* et *multicast* ainsi qu'une stratégie originale nommée *unregistered remote provider*. Cette dernière est une attaque que nous avons découverte et qui se base sur un comportement qui n'est pas ou qui est mal défini dans les spécifications du protocole NDN, à savoir le traitement d'un paquet *Data* qui vient d'une *Face* qui ne possède pas de route pour le nom du paquet *Data* reçu. Les résultats montrent que pour le premier type d'attaque, la stratégie *best-route* permet de mieux contenir l'attaque que la stratégie *multicast*. Ce type d'attaque, en plus d'affecter les utilisateurs, affecte aussi dans en bonne proportion le fournisseur de contenus qui va recevoir bien plus de paquets *Interest* qu'en temps normal ainsi que le routeur d'accès où les utilisateurs sont « connectés ». Dans le cas de l'attaque *unregistered remote provider*, celle-ci est invisible pour le fournisseur de contenus car l'attaquant se contente d'envoyer des paquets *Data* de façon opportuniste (dans la variante utilisée). Cette attaque a aussi un effet très important sur le routeur sur lequel l'attaquant est connecté. Mais semble moins impacter le routeur d'accès par rapport aux autres attaques. Cette attaque est très simple à mettre en œuvre car il suffit de se « connecter » à un routeur pour pouvoir la réaliser. Mais il est assez simple de se protéger contre elle et deux propositions basées sur les informations contenues dans la PIT ou la FIB permettent de l'empêcher. Pour le cas des autres attaques basées sur la coopération il est plus difficile de les prévenir car leur comportement est similaire aux utilisateurs légitimes. Néanmoins les prérequis sont plus complexes à réaliser pour pouvoir mettre en place ce type d'attaque. Pour finir cette étude, une analyse en composantes principales à été réalisée sur la base de nos résultats et montre qu'il est possible de détecter la CPA mais aussi d'identifier les deux types d'attaque utilisées.

Des propositions d'adaptation de protocoles existants pour NDN ont été faites mais la question de l'interopérabilité entre les deux types de réseau n'est que rarement évoquée. On distingue deux types de solutions pour permettre l'interopérabilité : des *gateways* permettant d'adapter un protocole ou la prise en charge native. Nous pensons qu'il est trop tôt pour une prise en charge native car cela demande un travail conséquent pour adapter les applications existantes et cela demande aussi la mise en place d'un réseau compatible de bout en bout. Les *gateways* quant à elles ne nécessitent qu'un réseau partiel sous forme d'îlot. Nous pensons que les *gateways* sont plus adaptées pour une première phase de déploiement, c'est pourquoi nous avons choisi d'explorer ce type de solution. Notre but est de développer une paire de *gateways* qui utilise au mieux les fonctionnalités du réseau NDN. Une implémentation de *gateways* HTTP existe déjà mais celle-ci présente des lacunes qui ne permettent pas un usage efficace du réseau. Une autre implémentation de *gateways* pour TCP a été proposée, celle-ci est certainement la plus intéres-

sante de par sa versatilité mais cela fait aussi sa faiblesse car le transport de paquets bas niveau ne permet pas une mutualisation des contenus (cache). Dans cette contribution, trois grands axes sont traités, le nommage, la mutualisation des contenus et l'évaluation de la solution. Le protocole d'adaptation développé se base sur trois étapes. D'abord, la notification de la requête qui permet de notifier au réseau qu'une nouvelle requête doit être satisfaite, cette étape est réalisée par la *gateway* d'entrée. Ensuite, une ou plusieurs *gateway* de sortie vont demander le contenu précis de cette requête et la satisfaire en communiquant sur le réseau IP. En parallèle, la *gateway* d'entrée demandera d'elle-même les morceaux de la réponse puisque aucune connexion n'est établie. Pour la mutualisation des contenus, nous nous sommes concentré sur les contenus statiques, qui sont les plus à même d'être mis en cache. Ainsi, sans traitement préalable, notre évaluation montre que le cache ne peut être utilisé que pour un même utilisateur. Cela s'explique par l'unicité des requêtes HTTP des utilisateurs à cause de certains champs définis dans son entête avec par exemple « cookies ». En analysant l'occurrence de ces différents champs et leur impact sur le résultat des requêtes, nous avons identifié ceux qui n'affectent pas la réponse et utilisons ce résultat pour modifier en amont (au niveau de la *gateway* d'entrée) ces champs pour les contenus statiques. Cette modification de l'entête HTTP permet une mutualisation jusqu'à 37,8% si l'on ne modifie pas le champ « cookie » (au lieu de 0%) et jusqu'à 78,3% dans l'affirmative. Notre évaluation montre que nos *gateways* sont relativement performantes et n'ajoutent qu'un délai fixe pour chaque requête traitée. Elles sont relativement efficaces avec un taux d'échec faible (<1%) qui arrive notamment sur les sites les plus distants. L'îlot NDN montre de bonnes performances par rapport à IP lorsque le cache est utilisé. L'écart s'accroît logiquement en faveur de NDN pour les continents éloignés (Océanie, Asie, Amérique du sud). Une simulation de trafic utilisateur montre que les gains attendus du cache NDN sur la consommation de bande passante sont de 18,8% et 25,1% selon la distribution de popularité des contenus (lois Zipf de paramètre 1,2 ou 1,5).

Beaucoup de solutions permettent le déploiement de réseaux ICN sur des réseaux IP existants. La majorité se base sur des contrôleurs SDN pour combler les lacunes des réseaux IP et router correctement les paquets ICN. Beaucoup moins de solutions utilisent la virtualisation pour déployer un réseau ICN à côté d'un réseau IP. En plus de la virtualisation, le paradigme NFV apporte des concepts issus du *cloud computing* qui vont simplifier le déploiement de réseaux virtuels. C'est dans cette optique que notre travail se positionne, nous proposons ainsi comme dernière contribution une architecture à base de microservices ainsi qu'un manager simplifié pour le déploiement d'un réseau NDN. Notre contribution est à notre connaissance la première architecture de ce type pour déployer un réseau ICN. Nous avons développé sept microservices différents dont cinq d'entre eux représentent les fonctions principales d'un routeur NDN : CS, PIT, FIB, stratégie, et pipeline de paquets, ainsi que deux autres fonctions que nous jugeons utiles pour la sécurité du réseau : vérification de signature et filtre de noms. Ces fonctions peuvent être agencées dans n'importe quel ordre tant que les cardinalités associées à chacune sont respectées. Il est ainsi possible de reproduire le comportement d'un nœud NFD tout en étant plus souple et plus performant. Notre manager est en charge de la gestion du réseau et de son bon fonctionnement. Pour ce faire, celui-ci maintient un modèle du réseau et stocke des informations relatives à chaque nœud. Il est capable de créer et détruire des microservices et de les lier dynamiquement, ce qui permet d'avoir un réseau qui s'adapte à la volée aux contraintes qui lui sont imposées. Le manager est ainsi capable de mettre à l'échelle les fonctions qu'il juge limitantes ou encore de placer dynamiquement des fonctions de sécurité si un problème à été identifié sur le réseau. Il gère aussi la propagation de route de manière centralisée de telle sorte qu'aucun protocole de routage interne n'est nécessaire. Les modules développés sont tous plus rapides que NFD, qu'ils

soient chaînés ou non, mais montrent une efficacité réduite par rapport à un nœud monolithique (un résultat attendu). Nous avons effectué un test de mise à l'échelle de la PIT, la fonction la plus coûteuse du routeur, qui montre un gain réel. Nous avons aussi pu mettre en évidence les capacités du manager à placer une contre-mesure contre une attaque CPA en déployant une fonction de vérification de signature puis de filtrage.

2 Perspectives de recherche

Un îlot NDN constitué de fonctions réseaux virtualisées a de nombreux avantages, comme nous avons pû le montrer dans ce manuscrit. En revanche, l'architecture x86 n'est pas optimale pour réaliser les traitements rapides et peu complexes que constituent certaines fonctions réseau comme le routage des paquets. Ainsi, afin de gagner en performances et surtout en efficacité, notre architecture gagnerait à intégrer le concept de switchs programmables via, par exemple, le langage P4, sachant que des travaux récents [Sig18] ont amorcé cette piste de recherche.

Le développement d'une architecture à base de microservices est une tâche complexe et chronophage. Celle-ci étant notre dernière contribution, quelques concessions ont dû être faites. Même si la majorité des concepts sont bien établis et fonctionnels, certains n'ont pas été exploités à leur plein potentiel par manque de temps. C'est notamment le cas des contre-mesures à l'attaque CPA. En effet, le manager a montré sa capacité à déployer dynamiquement des microservices à des fins de sécurité tels que le module de vérification de signature et le pare-feu. Cependant, pour lutter au mieux contre ce type d'attaque par déni de service, il convient d'effectuer le filtrage au plus proche de la source. Il faudrait donc enrichir le manager avec une stratégie de défense plus évoluée visant à déplacer de proche en proche les fonctions réseau servant de contre-mesures vers la source de l'attaque.

Un autre point important est la prise en compte des réseaux voisins de notre îlot NDN virtualisé. En effet, notre architecture doit pouvoir être à l'écoute des protocoles de routage externes (comme BGP dans le monde IP) afin de prendre en compte les annonces de routes de ses voisins. Ainsi, de nouvelles fonctions devront être conçues et placées en bordure de réseau pour transmettre les annonces de routes (par exemple reçues via NLSR) au manager qui se chargera ensuite de répercuter les changements sur les routeurs internes. D'autres microservices peuvent bien sûr être proposés. Un exemple serait une fonction permettant de réaliser un service de comptage (*accounting*) des contenus délivrés car une limite actuelle de l'architecture NDN pouvant freiner son adoption est que les fournisseurs de contenus ne sont plus capables de comptabiliser l'intégralité des accès à leurs contenus. Les FAI pourraient d'ailleurs monnayer la mise en place d'un tel service. Pour finir avec les microservices, une autre amélioration serait de trouver une fonction pour la mise à l'échelle de la PIT côté sortie (son but serait de réduire le nombre de connexions comme le fait la PIT), cette fonction est essentielle pour une mise à l'échelle efficace et pour ne pas impacter les autres modules. Nous croyons au potentiel de notre architecture à base de microservices qui, outre son originalité d'être la seule à pousser le concept des microservices jusqu'au routage du protocole (d'autres existent mais pour de l'applicatif comme NFN ou NFaaS), pourrait inciter la communauté ICN à l'adopter pour accélérer l'innovation autour de NDN.

Enfin, tous ces travaux resteraient vains sans un déploiement réel impliquant des utilisateurs. Nous comptons ainsi connecter notre îlot NDN virtualisé au testbed global NDN [23]. La passerelle HTTP/NDN permettra ainsi aux contenus web d'être véhiculés sur le réseau NDN et offrira un point d'entrée simple aux utilisateurs. Cela devra permettre de conduire des études plus réalistes

23. `https://named-data.net/ndn-testbed/`

de par la quantité et la qualité du trafic transitant sur le réseau NDN. Le fait d'impliquer des utilisateurs finaux permettra également de réaliser des études sur la qualité d'expérience afin de mesurer la capacité réelle d'adoption de ces nouveaux protocoles. Pour faire le lien avec le premier paragraphe de cette thèse, le testbed NDN ainsi augmenté pourrait former un noyau viable qui serait le point de départ d'un futur Internet basé sur le paradigme ICN, comme le fut ARPAnet en son temps pour IP.

Productions de la thèse

1 Publications scientifiques

Les contributions présentées dans ce mémoire ont donné lieu à diverses publications scientifiques. Ainsi, quatre articles scientifiques ont été publiés dans des conférences avec actes très sélectives :

- **Server-side performance evaluation of NDN**, publié à ACM ICN 2016 [MCF16b], correspond aux résultats sur les performances du routeur NFD et d'un fournisseur de contenu, ainsi que les améliorations proposées pour améliorer son efficacité présentés dans le chapitre 3.
- **Content Poisoning in Named Data Networking : Comprehensive Characterization of real Deployment**, publié à IFIP/IEEE IM 2017 [NMD+17], correspond aux résultats sur les différentes attaques et leur impact sur les différents éléments du réseau ainsi que l'ACP présentés dans le chapitre 4.
- **Leveraging NFV for the deployment of NDN : Application to HTTP traffic transport**, publié à IFIP/IEEE NOMS 2018 [MEAM+18], correspond à des expériences préliminaires réaliser avec les *gateways* et présente succinctement leur schéma de nommage ainsi que le *testbed* utilisé.
- **μNDN : an Orchestrated Microservice Architecture for Named Data Networking**, publié à ACM ICN 2018 [MCF18], correspond à la définition et l'utilisation des microservices et de leur manager ainsi que les premières expérimentations présentés dans le chapitre 6.

Deux articles de revue sont à ce jour en cours d'évaluation :

- **An Orchestrated NDN Virtual Infrastructure transporting Web Traffic : Design, Implementation and First Experiments with Real End-Users** soumis à IEEE Communications Magazine, actuellement en révision majeure, synthétise les principales contributions du projet ANR DOCTOR auxquelles cette thèse a contribué.
- **Bringing the Web on NDN : from the design of a mapping protocol to the evaluation of a gateway architecture** soumis à Elsevier Computer Networks reprend l'ensemble des résultats présentés dans le chapitre 5.

En plus des articles scientifiques, plusieurs démonstrations ont été sélectionnées et présentées dans des conférences avec actes, donnant lieu à des publications courtes :

- **A virtualized and monitored NDN infrastructure featuring a NDN/HTTP gateway**, publié dans la session des démonstrations de ACM ICN 2016 [MEAM+16], correspond à un premier déploiement des *gateways* HTTP/NDN en utilisant le testbed défini par le projet DOCTOR.
- **PIT matching from unregistered remote Faces : a critical NDN vulnerability**, publié dans la session des démonstrations de ACM ICN 2016 [MCF16a], correspond au premier scénario d'attaque présenté dans le chapitre 4 ainsi que les différentes propositions

de patchs pour y remédier.

— **Implementation of Content Poisoning Attack Detection and Reaction in Virtualized NDN Networks**, publié dans la session des démonstrations de ICIN 2018 [MAD+18], correspond à une mise en application de la détection de la CPA et la mise en place d'une contre-mesure grâce au monitoring et à l'orchestration d'un réseau virtuel.

2 Développements logiciels

Les travaux réalisés durant cette thèse revêtent volontairement un aspect pratique. Trois logiciels ont ainsi été créés et diffusés en open-source pour servir les futurs travaux de recherche sur les ICN.

— **NDNperf** est un outil similaire à iperf et permettant de faire des mesures de débit NDN dans différentes conditions. Il a été utilisé pour réaliser les expériences présentées dans le chapitre 2 mais aussi quelques unes dans les autres chapitres. Il est disponible dans le dépôt suivant : `https://github.com/Kanemochi/ndnperf`.

— **Les *gateways* HTTP/NDN** sont deux programmes permettant la traduction HTTP/NDN et plus généralement le transport optimisé de HTTP sur NDN, tel que décrit dans le chapitre 5. Elles sont disponibles dans le dépôt suivant qui inclut également un serveur web NDN natif : `https://github.com/Kanemochi/NDN-HTTP-Gateway` .

— **μNDN**, un ensemble de sept microservices orchestrés pour NDN. En plus des microservices et du manager présentés dans le chapitre 6 est inclus l'interface web de management du réseau NDN virtualisé. L'ensemble est disponible dans le dépôt suivant : `https://github.com/Kanemochi/NDN-microservices`.

Table des figures

Table des tableaux

Table des listes

Bibliographie

[ACG+13] Gergely Acs, Mauro Conti, Paolo Gasti, Cesar Ghali, and Gene Tsudik. Cache privacy in named-data networking. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pages 41–51. IEEE, 2013.

[ADI+12] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Borje Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7), 2012.

[AHZ15] Eslam G AbdAllah, Hossam S Hassanein, and Mohammad Zulkernine. A survey of security attacks in information-centric networking. *IEEE Communications Surveys & Tutorials*, 17(3) :1441–1454, 2015.

[AKRS11] S Arianfar, T Koponen, B Raghavan, and S Shenker. On preserving privacy in information-centric networks. In *Proc of SIGCOMM Workshop on ICN*, 2011.

[AMM+13] Alexander Afanasyev, Priya Mahadevan, Ilya Moiseenko, Ersin Uzun, and Lixia Zhang. Interest flooding attack and countermeasures in named data networking. In *IFIP Networking Conference, 2013*, pages 1–9. IEEE, 2013.

[ASZ+14] Alexander Afanasyev, Junxiao Shi, Beichuan Zhang, Lixia Zhang, Ilya Moiseenko, Yingdi Yu, Wentao Shang, Yi Huang, Jerald Paul Abraham, Steve DiBenedetto, et al. Nfd developer's guide. Technical report, Technical Report NDN-0021, NDN, 2014.

[ASZ+15] Alexander Afanasyev, Junxiao Shi, Beichuan Zhang, Lixia Zhang, Ilya Moiseenko, Yingdi Yu, Wentao Shang, Yanbiao Li, Spyridon Mastorakis, Yi Huang, Jerald Paul Abraham, Steve DiBenedetto, Chengyu Fan, Christos Papadopoulos, Davide Pesavento, Giulio Grassi, Giovanni Pau, Hang Zhang, Tian Song, Haowei Yuan, Hila Ben Abraham, Patrick Crowley, Syed Obaid Amin, Vince Lehman, and Lan Wang. Nfd developer's guide. Technical Report NDN-0021, Revision 4, NDN, May 2015.

[AZ15] Muhammad Aamir and Syed Mustafa Ali Zaidi. Denial-of-service in content centric (named data) networking : a tutorial and state-of-the-art survey. *Security and Communication Networks*, 8(11) :2037–2059, 2015.

[BDCBM13] Giuseppe Bianchi, Andrea Detti, Alberto Caponi, and Nicola Blefari Melazzi. Check before storing : What is the performance price of content integrity verification in lru caching ? *ACM SIGCOMM Computer Communication Review*, 43(3) :59–67, 2013.

[Bra09] Van Jacobson, Diana K. Smetters, Nicholas H. Briggs, James D. Thornton, Michael F. Plass, Rebecca L. Braynard. Networking named content. Technical report, Palo Alto Research Center, CA, USA, 2009.

[Bur13] Jeff Burke. Video streaming over named data networking. *E-LETTER*, 2013.

[CCGT13] Alberto Compagno, Mauro Conti, Paolo Gasti, and Gene Tsudik. Poseidon : Mitigating interest flooding ddos attacks in named data networking. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 630–638. IEEE, 2013.

[CCW+12] Margaret Chiosi, Don Clarke, Peter Willis, Andy Reid, James Feger, Michael Bugenhagen, Waqar Khan, Michael Fargano, Chunfeng Cui, Hui Deng, et al. Network functions virtualisation : An introduction, benefits, enablers, challenges and call for action. In *SDN and OpenFlow World Congress*, volume 48. sn, 2012.

[CDCKU13] Abdelberi Chaabane, Emiliano De Cristofaro, Mohamed Ali Kaafar, and Ersin Uzun. Privacy in content-oriented networking : Threats and countermeasures. *ACM SIGCOMM Computer Communication Review*, 43(3) :25–33, 2013.

[CfNoICN] Design Choices, Differences for NDN, and CCNx 1.0 Implementations of Information-Centric Networking. `https://icnrg.github.io/draft-icnrg-harmonization/draft-icnrg-harmonization-00.html`.

[CGM12] Giovanna Carofiglio, Massimo Gallo, and Luca Muscariello. ICP : Design and evaluation of an Interest control protocol for content-centric networking. In *IEEE INFOCOM Workshops*, pages 304–309, 2012.

[CGT13] Mauro Conti, Paolo Gasti, and Marco Teoli. A lightweight mechanism for detection of cache pollution attacks in named data networking. *Computer Networks*, 57(16) :3178–3191, 2013.

[Cha] Information-Centric Networking (ICN) Research Challenges. `http://www.rfc-editor.org/rfc/rfc7927.txt`.

[Cis18] Cisco. Cisco Visual Networking Index : Forecast and Trends, 2017–2022. `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf`, 2018.

[CK74] V. Cerf and R. Kahn. A protocol for packet network intercommunication. *IEEE Transactions on Communications*, 22(5) :637–648, May 1974.

[CRMa13] Carlos M.S. Cabral, Christian Esteve Rothenberg, and Maurício Ferreira Magalhães. Mini-ccnx : Fast prototyping for named data networking. In *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, ICN '13, pages 33–34, New York, NY, USA, 2013. ACM.

[DGCK08] Leiwen Deng, Yan Gao, Yan Chen, and Aleksandar Kuzmanovic. Pollution attacks and defenses for internet caching systems. *Computer Networks*, 52(5) :935–956, 2008.

[DLC+16] Kun Ding, Yun Liu, Hsin-Hung Cho, Han-Chieh Chao, and Timothy K Shih. Cooperative detection and protection for interest flooding attacks in named data networking. *International Journal of Communication Systems*, 29(13) :1968–1980, 2016.

[DM14] Namiot Dmitry and Sneps-Sneppe Manfred. On micro-services architecture. *International Journal of Open Information Technologies*, 2(9), 2014.

[DP16] Steve DiBenedetto and Christos Papadopoulos. Mitigating poisoned content with forwarding strategy. In *The third Workshop on Name-Oriented Mobility (NOM)*. IEEE, 2016.

[Duk08] Nandita Dukkipati. *Rate Control Protocol (RCP) : Congestion control to make flows complete quickly*. Citeseer, 2008.

[FLT+13] Seyed Kaveh Fayazbakhsh, Yin Lin, Amin Tootoonchian, Ali Ghodsi, Teemu Koponen, Bruce Maggs, K.C. Ng, Vyas Sekar, and Scott Shenker. Less pain, most of the gain : Incrementally deployable icn. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 147–158, New York, NY, USA, 2013. ACM.

[Fow14] Martin Fowler. Microservices a definition of this new architectural term, 2014.

[GFT+13] Pedro Henrique V Guimaraes, Lyno Henrique G Ferraz, Joao Vitor Torres, Diogo MF Mattos, P Murillo, F Andres, L Andreoni, E Martin, Igor D Alvarenga, Claudia SC Rodrigues, et al. Experimenting Content-Centric Networks in the future internet testbed environment. In *IEEE International Conference on Communications Workshops (ICC)*, pages 1383–1387. IEEE, 2013.

[GKS+18] Cenk Gündoğan, Peter Kietzmann, Thomas C Schmidt, Martine Lenders, Hauke Petersen, and Matthias Wählisch. Ndn, coap, and mqtt : A comparative measurement study in the iot. *arXiv preprint arXiv :1806.01444*, 2018.

[GKSW18a] Cenk Gündoğan, Peter Kietzmann, Thomas C Schmidt, and Matthias Wählisch. Hopp : Publish–subscribe for the constrained iot. 2018.

[GKSW18b] Cenk Gündoğan, Peter Kietzmann, Thomas C Schmidt, and Matthias Wählisch. Hopp : Robust and resilient publish-subscribe for an information-centric internet of things. *arXiv preprint arXiv :1801.03890*, 2018.

[GKSW18c] Cenk Gündoğan, Peter Kietzmann, Thomas C Schmidt, and Matthias Wählisch. Icn-lowpan : Header compression for the constrained iot. 2018.

[Gro13] Network Functions Virtualisation Industry Specification Group. Network Functions Virtualisation (NFV). White paper, ETSI, 2013.

[Gro14] Network Functions Virtualisation Industry Specification Group. Network Functions Virtualisation (NFV) ; Management and Orchestration. Group Specification GS NFV-MAN 001, ETSI, 2014.

[GTU14a] Cesar Ghali, Gene Tsudik, and Ersin Uzun. Needle in a haystack : Mitigating content poisoning in named-data networking. In *Proceedings of NDSS Workshop on Security of Emerging Networking Technologies (SENT)*, 2014.

[GTU14b] Cesar Ghali, Gene Tsudik, and Ersin Uzun. Network-layer trust in named-data networking. *ACM SIGCOMM Computer Communication Review*, 44(5) :12–19, 2014.

[GTUZ13] Paolo Gasti, Gene Tsudik, Ersin Uzun, and Lixia Zhang. Dos and ddos in named data networking. In *Computer Communications and Networks (ICCCN), 2013 22nd International Conference on*, pages 1–7. IEEE, 2013.

[HAA+13] AKM Hoque, Syed Obaid Amin, Adam Alyyan, Beichuan Zhang, Lixia Zhang, and Lan Wang. Nlsr : named-data link state routing protocol. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pages 15–20. ACM, 2013.

[HGJL15] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization : Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2) :90–97, 2015.

[Jac06] Van Jacobson. A new way to look at networking, google tech talk. `https://www.youtube.com/watch?v=oCZMoY3q2uM`, 2006.

[JMT18] Song Junghwan, Lee Munyoung, and Kwon Ted. Smic : Subflow-level multi-path interest control for information centric networking. In *Proceedings of the 5th ACM Conference on Information-Centric Networking (ACM ICN'18)*, 2018.

[JSB+09] Van Jacobson, Diana K Smetters, Nicholas H Briggs, Michael F Plass, Paul Stewart, James D Thornton, and Rebecca L Braynard. Voccn : voice-over content-centric networks. In *Proceedings of the 2009 workshop on Re-architecting the internet*, pages 1–6. ACM, 2009.

[JST+09] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.

[KCC+07] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 37, pages 181–192. ACM, 2007.

[KGZ15] Amin Karami and Manel Guerrero-Zapata. An anfis-based cache replacement method for mitigating cache pollution attacks in named data networking. *Computer Networks*, 80 :51–65, 2015.

[KNBY15] Dohyung Kim, Sunwook Nam, Jun Bi, and Ikjun Yeom. Efficient content verification in named data networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking*, pages 109–116. ACM, 2015.

[KP17] Michał Król and Ioannis Psaras. Nfaas : named function as a service. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pages 134–144. ACM, 2017.

[KSG+17] Kevin Khanda, Dilshat Salikhov, Kamill Gusmanov, Manuel Mazzara, and Nikolaos Mavridis. Microservice-based iot for smart buildings. In *Advanced Information Networking and Applications Workshops (WAINA), 2017 31st International Conference on*, pages 302–308. IEEE, 2017.

[Lau10] Tobias Lauinger. *Security & scalability of content-centric networking.* PhD thesis, TU Darmstadt, 2010.

[LC15] Yong Li and Min Chen. Software-defined network function virtualization : A survey. *IEEE Access*, 3 :2542–2553, 2015.

[LGP+13] Yaning Liu, Joost Geurts, Jean-Charles Point, Stefan Lederer, Benjamin Rainer, Christopher Muller, Christian Timmerer, and Hermann Hellwagner. Dynamic adaptive streaming over ccn : A caching and overhead analysis. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3629–3633. IEEE, 2013.

[LLZM14] Zhuo Li, Kaihua Liu, Yang Zhao, and Yongtao Ma. Mapit : an enhanced pending interest table for ndn with mapping bloom filter. *IEEE Communications Letters*, 18(11) :1915–1918, 2014.

[LTV16] Alessandra Levcovitz, Ricardo Terra, and Marco Tulio Valente. Towards a technique for extracting microservices from monolithic enterprise systems. *arXiv preprint arXiv :1605.03175*, 2016.

[LWD10] Chao Liu, Ryen W White, and Susan Dumais. Understanding web browsing behaviors through weibull analysis of dwell time. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 379–386. ACM, 2010.

[MAD+18] Hoang Long Mai, Messaoud Aouadj, Guillaume Doyen, Daishi Kondo, Xavier Marchal, Thibault Cholez, Edgardo Montes De Oca, and Wissam Mallouli. Implementation of Content Poisoning Attack Detection and Reaction in Virtualized NDN Networks. In *ICIN 2018 - 21st Conference on Innovation in Clouds, Internet and Networks*, page 3, Paris, France, February 2018.

[MAGO16] Milad Mahdian, Somaya Arianfar, Jim Gibson, and Dave Oran. Mircc : Multipath-aware icn rate-based congestion control. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 1–10. ACM, 2016.

[MAM+16] Xavier Marchal, Moustapha El Aoun, Bertrand Mathieu, Wissam Mallouli, Thibault Cholez, Guillaume Doyen, Patrick Truong, Alain Ploix, and Edgardo Montes de Oca. A virtualized and monitored NDN infrastructure featuring a NDN/HTTP gateway. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking, ICN '16, Kyoto, Japan*, pages 225–226. ACM, 2016.

[MAR+14] Joao Martins, Mohamed Ahmed, Costin Raiciu, Vladimir Olteanu, Michio Honda, Roberto Bifulco, and Felipe Huici. Clickos and the art of network function virtualization. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, pages 459–473. USENIX Association, 2014.

[MCF16a] Xavier Marchal, Thibault Cholez, and Olivier Festor. PIT matching from unregistered remote Faces : a critical NDN vulnerability. 3rd ACM Conference on Information-Centric Networking (ACM-ICN'16), September 2016. Poster.

[MCF16b] Xavier Marchal, Thibault Cholez, and Olivier Festor. Server-side performance evaluation of NDN. In *3rd ACM Conference on Information-Centric Networking (ACM-ICN'16)*, pages 148 – 153, Kyoto, Japan, September 2016. ACM SIGCOMM, ACM.

[MCF18] Xavier Marchal, Thibault Cholez, and Olivier Festor. μNDN : an Orchestrated Microservice Architecture for Named Data Networking. In *ACM-ICN'18 - 5th ACM Conference on Information-Centric Networking*, page 12, Boston, United States, September 2018.

[MEAM+16] Xavier Marchal, Moustapha El Aoun, Bertrand Mathieu, Wissam Mallouli, Thibault Cholez, Guillaume Doyen, Patrick Truong, Alain Ploix, and Edgardo Montes De Oca. A virtualized and monitored NDN infrastructure featuring a NDN/HTTP gateway. 3rd ACM Conference on Information-Centric Networking (ACM-ICN'16), September 2016. Poster.

[MEAM+18] Xavier Marchal, Moustapha El Aoun, Bertrand Mathieu, Thibault Cholez, Guillaume Doyen, Wissam Mallouli, and Olivier Festor. Leveraging NFV for the deployment of NDN : Application to HTTP traffic transport. In *NOMS 2018 - IEEE/IFIP Network Operations and Management Symposium*, page 5, Taipei, Taiwan, April 2018. IEEE.

[MMR+13] Anil Madhavapeddy, Richard Mortier, Charalampos Rotsos, David Scott, Balraj Singh, Thomas Gazagnaire, Steven Smith, Steven Hand, and Jon Crowcroft. Unikernels : Library operating systems for the cloud. *SIGPLAN Not.*, 48(4) :461–472, March 2013.

[MND+16] Hoang Long Mai, Ngoc Tan Nguyen, Guillaume Doyen, Alain Ploix, and Remi Cogranne. On the readiness of ndn for a secure deployment : The case of pending interest table. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 98–110. Springer, 2016.

[MND+18] Hoang Long Mai, Tan Nguyen, Guillaume Doyen, Rémi Cogranne, Wissam Mallouli, Edgardo Montes de Oca, and Olivier Festor. Towards a security monitoring plane for named data networking and its application against content poisoning attack. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9. IEEE, 2018.

[MO16] Ilya Moiseenko and Dave Oran. Tcp/icn : Carrying tcp over content centric and named data networks. In *2016 conference on 3rd ACM Conference on Information-Centric Networking*, pages 112–121. ACM, 2016.

[MSG+16] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network function virtualization : State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 18(1) :236–262, 2016.

[MSO14] Ilya Moiseenko, Mark Stapp, and David Oran. Communication patterns for web interaction in named data networking. In *1st international conference on Information-centric networking*, pages 87–96. ACM, 2014.

[MSUW15] Marc Mosko, Ignacio Solis, Ersin Uzun, and Christopher Wood. Ccnx 1.0 protocol architecture. *Palo Alto Research Center. url http ://ccnx. org/-pubs/ICN_ CCN_ Protocols. pdf*, 2015.

[MZS+13] Abedelaziz Mohaisen, Xinwen Zhang, Max Schuchard, Haiyong Xie, and Yongdae Kim. Protecting access privacy of cached contents in information centric networks. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 173–178. ACM, 2013.

[NCDR15] Tan N Nguyen, Rémi Cogranne, Guillaume Doyen, and Florent Retraint. Detection of interest flooding attacks in named data networking using hypothesis testing. In *Information Forensics and Security (WIFS), 2015 IEEE International Workshop on*, pages 1–6. IEEE, 2015.

[NDN] NDN Team : http ://named-data.net/.

[Net] Hybrid Information-Centric Networking. `https://datatracker.ietf.org/doc/draft-muscariello-intarea-hicn/`.

[NH12] Nonhlanhla Ntuli and Sunyoung Han. Detecting router cache snooping in named data networking. In *ICT Convergence (ICTC), 2012 International Conference on*, pages 714–718. IEEE, 2012.

[NMD+17] Tan Nguyen, Xavier Marchal, Guillaume Doyen, Thibault Cholez, and Rémi Cogranne. Content Poisoning in Named Data Networking : Comprehensive Characterization of real Deployment. In *15th IFIP/IEEE International Symposium on Integrated Network Management (IM2017)*, pages 72–80, Lisbon, Portugal, May 2017.

[NST13] Xuan Nam Nguyen, D. Saucez, and T. Turletti. Efficient caching in content-centric networks using openflow. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 67–68, April 2013.

[ORS12] Sara Oueslati, James Roberts, and Nada Sbihi. Flow-aware traffic control for a Content-Centric Network. In *Proceedings of IEEE INFOCOM 2012*, pages 2417–2425, 2012.

[PGL+16] D. Perino, M. Gallo, R. Laufer, Z. B. Houidi, and F. Pianese. A programmable data plane for heterogeneous nfv platforms. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 77–82, April 2016.

[Pro] Named Data Networking Next Phase (NDN-NP) Proposal. `http://www.caida.org/funding/ndn-np/ndn-np_proposal.xml`.

[QNP+15] Xiuquan Qiao, Guoshun Nan, Yue Peng, Lei Guo, Jingwen Chen, Yunlei Sun, and Junliang Chen. Ndnbrowser : An extended web browser for named data networking. *Journal of Network and Computer Applications*, 50 :134–147, 2015.

[QNT+14] Xiuquan Qiao, Guoshun Nan, Wei Tan, Lei Guo, Junliang Chen, Wei Quan, and Yukai Tu. Ccnxtomcat : An extended web server for content-centric networking. *Computer Networks*, 75 :276–296, 2014.

[QXG+14] Wei Quan, Changqiao Xu, Jianfeng Guan, Hongke Zhang, and Luigi Alfredo Grieco. Scalable name lookup with adaptive prefix bloom filter for named data networking. *IEEE Communications Letters*, 18(1) :102–105, 2014.

[RCA+17] R. Ravindran, A. Chakraborti, S. O. Amin, A. Azgin, and G. Wang. 5g-icn : Delivering icn services over 5g using network slicing. *IEEE Communications Magazine*, 55(5) :101–107, May 2017.

[Ric17] Chris Richardson. Microservice architecture, 2017.

[RR11] Dario Rossi and Giuseppe Rossini. Caching performance of content centric networks under multi-path routing (and more). *Relatório técnico, Telecom ParisTech*, 2011.

[RRAG14] Igor Ribeiro, Antonio Rocha, Celio Albuquerque, and Flavio Guimaraes. On the possibility of mitigating content pollution in content-centric networking. In *Local Computer Networks (LCN), 2014 IEEE 39th Conference on*, pages 498–501. IEEE, 2014.

[RTKR18] A. Rahman, D. Trossen, D. Kutscher, and R. Ravindran. Deployment Considerations for Information-Centric Networking. Internet-Draft draft-irtf-icnrg-deployment-guidelines-00, Internet Research Task Force, March 2018. Work in Progress.

[SBMD+13] Stefano Salsano, Nicola Blefari-Melazzi, Andrea Detti, Giacomo Morabito, and Luca Veltri. Information centric networking over sdn and openflow : Architectural aspects and experiments on the ofelia testbed. *Computer Networks*, 57(16) :3207–3221, 2013.

[Sem] CCNx Semantics. `https://datatracker.ietf.org/doc/draft-irtf-icnrg-ccnxsemantics/`.

[SFG+14] Yi Sun, Seyed Kaveh Fayaz, Yang Guo, Vyas Sekar, Yun Jin, Mohamed Ali Kaafar, and Steve Uhlig. Trace-driven analysis of icn caching algorithms on video-on-demand workloads. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, pages 363–376, New York, NY, USA, 2014. ACM.

[Sig18] Salvatore Signorello. *A multifold approach to address the security issues of stateful forwarding mechanisms in Information-Centric Networks. (Une approche multidimensionnelle pour aborder les problèmes de sécurité des mécanismes d'acheminement à états dans les réseaux orientés contenus)*. PhD thesis, University of Lorraine, Nancy, France, 2018.

[SKST14] Manolis Sifalakis, Basil Kohler, Christopher Scherb, and Christian Tschudin. An information centric network for computing the distribution of computations. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, pages 137–146. ACM, 2014.

[SMA+17] Mauro Sardara, Luca Muscariello, Jordan Augé, Marcel Enguehard, Alberto Compagno, and Giovanna Carofiglio. Virtualized icn (vicn) : Towards a unified network virtualization framework for icn experimentation. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, ICN '17, pages 109–115, New York, NY, USA, 2017. ACM.

[SMC18] Mauro Sardara, Luca Muscariello, and Alberto Compagno. A transport layer and socket api for (h) icn : Design, implementation and performance analysis. In *Proceedings of the 5th ACM Conference on Information-Centric Networking (ACM ICN'18)*, 2018.

[SNOS13] Won So, Ashok Narayanan, David Oran, and Mark Stapp. Named Data Networking on a router : forwarding at 20gbps and beyond. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 495–496. ACM, 2013.

[STC+13] Wentao Shang, Jeff Thompson, Meki Cherkaoui, Jeff Burkey, and Lixia Zhang. Ndn. js : A javascript client library for named data networking. In *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*, pages 399–404. IEEE, 2013.

[STPP15] Vasilis Sourlas, Leandros Tassiulas, Ioannis Psaras, and George Pavlou. Information resilience through user-assisted caching in disruptive content-centric networks. In *IFIP Networking Conference (IFIP Networking), 2015*, pages 1–9. IEEE, 2015.

[SWS15] Hani Salah, Julian Wulfheide, and Thorsten Strufe. Coordination supports security : A new defence mechanism against interest flooding in ndn. In *Local Computer Networks (LCN), 2015 IEEE 40th Conference on*, pages 73–81. IEEE, 2015.

[TBB+15] Giovanni Toffetti, Sandro Brunner, Martin Blöchlinger, Florian Dudouet, and Andrew Edmonds. An architecture for self-managing microservices. In *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*, AIMC '15, pages 19–24, New York, NY, USA, 2015. ACM.

[TZLZ13] Jianqiang Tang, Zhongyue Zhang, Ying Liu, and Hongke Zhang. Identifying interest flooding in named data networking. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*, pages 306–310. IEEE, 2013.

[UNO16] Takanori Ueda, Takuya Nakaike, and Moriyoshi Ohara. Workload characterization for microservices. In *Workload Characterization (IISWC), 2016 IEEE International Symposium on*, pages 1–10. IEEE, 2016.

[vAK15] Niels LM van Adrichem and Fernando A Kuipers. Ndnflow : software-defined named data networking. In *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pages 1–5. IEEE, 2015.

[VC] Carl Sunshine Vinton Cerf, Yogen Dalal. Rfc 675 : Specification of internet transmission control program. `https://tools.ietf.org/html/rfc675`.

[VSKS13] Markus Vahlenkamp, Fabian Schneider, Dirk Kutscher, and Jan Seedorf. Enabling information centric networking in ip networks using sdn. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–6. IEEE, 2013.

[WBW+12] Sen Wang, Jun Bi, Jianping Wu, Xu Yang, and Lingyuan Fan. On adapting http protocol to content centric networking. In *7th International Conference on Future Internet Technologies*, pages 1–6. ACM, 2012.

[WCZ+14] Kai Wang, Jia Chen, Huachun Zhou, Yajuan Qin, and Hongke Zhang. Modeling denial-of-service against pending interest table in named data networking. *International Journal of Communication Systems*, 27(12) :4355–4368, 2014.

[WPM+13] Yi Wang, Tian Pan, Zhian Mi, Huichen Dai, Xiaoyu Guo, Ting Zhang, Bin Liu, and Qunfeng Dong. Namefilter : Achieving fast name lookup with low memory cost via applying two-stage bloom filters. In *INFOCOM, 2013 Proceedings IEEE*, pages 95–99. IEEE, 2013.

[WR16] Greg White and Greg Rutz. Content Delivery With Content Centric Networking. Technical report, Cable Television Laboratories, 02 2016.

[WSV13] Matthias Wählisch, Thomas C Schmidt, and Markus Vahlenkamp. Backscatter from the data plane–threats to stability and security in information-centric network infrastructure. *Computer Networks*, 57(16) :3192–3206, 2013.

[WZQ+13] Kai Wang, Huachun Zhou, Yajuan Qin, Jia Chen, and Hongke Zhang. Decoupling malicious interests from pending interest table to mitigate interest flooding attacks. In *Globecom Workshops (GC Wkshps), 2013 IEEE*, pages 963–968. IEEE, 2013.

[WZZ+13] Yi Wang, Yuan Zu, Ting Zhang, Kunyang Peng, Qunfeng Dong, Bin Liu, Wei Meng, Huichen Dai, Xin Tian, Zhonghu Xu, et al. Wire speed name lookup : A gpu-based approach. In *NSDI*, pages 199–212, 2013.

[XCCC12] Hongfeng Xu, Zhen Chen, Rui Chen, and Junwei Cao. Live streaming with content centric networking. In *Networking and Distributed Computing (ICNDC), 2012 Third International Conference on*, pages 1–5. IEEE, 2012.

[XLW+16] Yonghui Xin, Yang Li, Wei Wang, Weiyuan Li, and Xin Chen. A novel interest flooding attacks detection and countermeasure scheme in ndn. In *Global Communications Conference (GLOBECOM), 2016 IEEE*, pages 1–7. IEEE, 2016.

[XVS+14] George Xylomenos, Christopher N Ververidis, Vasilios A Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V Katsaros, George C Polyzos, et al. A survey of information-centric networking research. *IEEE Communications Surveys and Tutorials*, 16(2) :1024–1049, 2014.

[XWW12] Mengjun Xie, Indra Widjaja, and Haining Wang. Enhancing cache robustness for content-centric networking. In *INFOCOM, 2012 Proceedings IEEE*, pages 2426–2434. IEEE, 2012.

[YC11] Haowei Yuan and Patrick Crowley. Performance measurement of name-centric content distribution methods. In *Proceedings of the 2011 ACM/IEEE Seventh Symposium on Architectures for Networking and Communications Systems*, pages 223–224. IEEE Computer Society, 2011.

[YC13] Haowei Yuan and Patrick Crowley. Experimental evaluation of content distribution with ndn and http. In *INFOCOM, 2013 Proceedings IEEE*, pages 240–244. IEEE, 2013.

[YSC12] Haowei Yuan, Tian Song, and Patrick Crowley. Scalable NDN forwarding : Concepts, issues and principles. In *Computer Communications and Networks (ICCCN), 21st International Conference on*, pages 1–9. IEEE, 2012.

[ZAB+14] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. Named Data Networking. *ACM SIGCOMM Computer Communication Review*, 44(3) :66–73, 2014.

[ZLL13] Guoqiang Zhang, Yang Li, and Tao Lin. Caching in information centric networking : A survey. *Computer Networks*, 57(16) :3128–3141, 2013.

[ZLZ15] Meng Zhang, Hongbin Luo, and Hongke Zhang. A survey of caching mechanisms in information-centric networking. *IEEE Communications Surveys & Tutorials*, 17(3) :1473–1499, 2015.

Cette thèse utilise 123 références.

Abstract

Architectures and advanced functions for a progressive deployment of Information-Centric Networking

Internet historical protocols (TCP/IP) that were used to interconnect the very first computers are no longer suitable for the massive distribution of content that is now being made. New content-based network protocols (Information-Centric Networking) are currently being designed to optimize these exchanges by betting on a paradigm shift where content, rather than machines, are addressable across the Internet. However, such a change can only be made gradually and if all operational imperatives are met. Thus, this thesis aims to study and remove the main technological obstacles preventing the adoption of the NDN (Name Data Networking) protocol by operators by guaranteeing the security, performance, interoperability, proper management and automated deployment of an NDN network.

First, we evaluate the current performance of an NDN network thanks to a tool we made, named ndnperf, and observe the high cost for a provider delivering fresh content using this protocol. Then, we propose some optimizations to improve the efficiency of packet generation up to 6.4 times better than the default parameters.

Afterwards, we focus on the security of the NDN protocol with the evaluation of the content poisoning attack, known as the second more critical attack on NDN, but never truly characterized. Our study is based on two scenarios, with the usage of a malicious user and content provider, or by exploiting a flaw we found in the packet processing flow of the NDN router. Thus, we show the danger of this kind of attacks and propose a software fix to prevent the most critical scenario.

Thirdly, we are trying to adapt the HTTP protocol in a way so that it can be transported on an NDN network for interoperability purposes. To do this, we designed an adaptation protocol and developed two gateways that perform the necessary conversions so that web content can seamlessly enter or exit an NDN network. After describing our solution, we evaluate and improve it in order to make web content benefit from a major NDN feature, the in-network caching, and show up to 61.3% cache-hit ratio in synthetic tests and 25.1% in average for browsing simulations with multiple users using a Zipf law of parameter 1.5.

Finally, we propose a virtualized and orchestrated microservice architecture for the deployment of an NDN network following the Network Fonction Virtualization (NFV) paradigm. We developed seven microservices that represent either an atomic function of the NDN router or a new one for specific purposes. These functions can then be chained to constitute a full-fledged network. Our architecture is orchestrated with the help of a manager that allows it to take the full advantages of the microservices like scaling the bottleneck functions or dynamically change the topology for the current needs (an attack for example).

Our architecture, associated with our other contributions on performance, security and interoperability, allows a better and more realistic deployment of NDN, especially with an easier development of new features, a network running on standard hardware, and the flexibility allowed by this kind of architecture.

Keywords: network, virtualization, security, network management, Internet protocol suite

Résumé

Architectures et fonctions avancées pour le déploiement progressif de réseaux orientés contenus

Les protocoles historiques d'Internet (TCP/IP) qui servaient à interconnecter les tous premiers ordinateurs ne sont plus adaptés à la diffusion massive de contenus qui en est fait aujourd'hui. De nouveaux protocoles réseau centrés sur les contenus (Information-Centric Networking) sont actuellement conçus pour optimiser ces échanges en pariant sur un changement de paradigme où les contenus, plutôt que les machines sont adressables à l'échelle d'Internet. Cependant, un tel changement ne peut se faire que progressivement et si tous les impératifs opérationnels sont assurés. Ainsi, cette thèse a pour objectif d'étudier et de lever les principaux verrous technologiques empêchant l'adoption du protocole NDN (Name Data Networking) par les opérateur en garantissant la sécurité, les performances, l'interopérabilité, la bonne gestion et le déploiement automatisé d'un réseau NDN.

Dans un premier temps, nous évaluons les performances actuelles d'un réseau NDN à l'aide d'un outil de notre conception, ndnperf, et constatons le coût élevé pour un serveur utilisant ce protocole. Puis nous proposons un ensemble de solutions pour améliorer l'efficacité d'un serveur NDN pouvant être jusqu'à 6,4 fois plus efficient que les paramètres de base.

Ensuite nous nous intéressons à la sécurité de NDN à travers l'évaluation de l'attaque par empoisonnement de contenus, connue pour être critique mais jamais caractérisée. Cette étude se base sur deux scénarios, en utilisant un serveur et un client pour effectuer la pollution, ou en exploitant une faille dans le traitement des paquets au niveau du routeur. Nous montrons ainsi la dangerosité de l'attaque et proposons une correction de la faille la permettant.

Dans un troisième temps, nous cherchons à adapter le protocole HTTP pour qu'il puisse être transporté sur un réseau NDN à des fins d'interopérabilité. Pour ce faire, nous avons développé deux passerelles qui effectuent les conversions nécessaires pour qu'un contenu web puisse rentrer ou sortir d'un réseau NDN. Après avoir décrit notre solution, nous l'évaluons et l'améliorons afin de pouvoir bénéficier d'une fonctionnalité majeure de NDN, à savoir la mise en cache des contenus dans le réseau à hauteur de 61,3% lors de tests synthétiques et 25,1% lors de simulations de navigation avec plusieurs utilisateurs utilisant une loi Zipf de paramètre 1,5.

Pour finir, nous proposons une architecture à base de microservices virtualisés et orchestrés pour le déploiement du protocole NDN en suivant le paradigme NFV (Network Function Virtualization). Les sept microservices présentés reprennent soit une fonction atomique du routeur, soit proposent un nouveau service spécifique. Ces fonctions peuvent ensuite être chaînées pour constituer un réseau optimisé. Cette architecture est orchestrée à l'aide d'un manager qui nous permet de pleinement tirer parti des avantages des microservices comme la mise à l'échelle des composants les plus lents ou encore le changement dynamique de topologie en cas d'attaque.

Une telle architecture, associée aux contributions précédentes, permettrait un déploiement rapide du protocole NDN, notamment grâce à un développement facilité des fonctions, à l'exécution sur du matériel conventionnel, ou encore grâce à la flexibilité qu'offre ce type d'architecture.

Mots-clés: réseau, virtualisation, sécurité, gestion de réseau, suite des protocoles Internet