A Content-Centric Networking Node for a Realistic Efficient Implementation and Deployment

Wei You

N° d'ordre : 2014telb0312

Sous le sceau de l' Université européenne de Bretagne

Télécom Bretagne

En accréditation conjointe avec l'Ecole Doctorale Matisse

Co-tutelle avec Orange Labs

Ecole Doctorale – MATISSE

A Content-Centric Networking Node for a Realistic Efficient Implementation and Deployment

---

Thèse de Doctorat

Mention : Informatique

Présentée par **Wei You**

Département : Informatique

Laboratoire : IRISA Pôle (uniquement si Lab-STICC) :

Directeur de thèse : Annie Gravey

Soutenue le 20 janvier 2014
(Seulement si date connue)

Acknowledgement

This three years research experience will be an important period in my life. Without the help and support from many people around me, there would be no possible that these work and this thesis can be finished.

First of all I would like to thank my parents for their support and understanding at all the time for that I studied in a country far away from home .

I would also like to thank Bertrand Mathieu, my supervisor at Orange Labs, who opened me a door of the CCN world. His advices, knowledge and guidance are invaluable. Without his help and support, I would not have finished this thesis. I also very appreciate that he left me a lot of freedom for carrying the research work on.

I grateful acknowledge my academic supervisor, Gwendal Simon, for his help. His academic knowledge, research spirit and his rigour encourage me a lot. I also want to thank him for his reviews of the papers that we finished together.

I would also want to give my thanks to Prosper Chemouil, my director at Orange Labs, and to Annie Gravey, my director at Télécom Bretagne, that they offered me this precious opportunity to work both at Orange Labs and at Télécom Bretagne.

Many thanks to all the colleagues that I worked or cooperated with: Patrick Truong, Jean-François Peltier, Jean-Louis Simon, Jean-Yves Mazeas, Tomas Morin and David Fahed from Orange Labs MOVI team, and Zhe Li, Jiayi Liu, Yaning Liu and Fen Zhou from Télécom Bretagne Info department.

Also I want to thank my previous team manager Myriam Vialat and current team manager Maria Luisa Guerra Feliz De Vargas, for their great support.

At last, I would like to thank everyone again who helped and supported me during these three years for the realization of my thesis.

Abstract

The current IP based Internet architecture is designed in 70s. With the technologies development and the Internet usage evolution, this design becomes heavier and heavier and less efficient for the Internet services, especially for the content delivery, which generates the main part of current networking traffic. Facing this shortcoming, many research forces propose an Information-Centric Networking (ICN) paradigm, which focus on an information-oriented networking architecture. Following this direction, Van Jacobson and his PARC team proposed the Content-Centric Network (CCN) solution in 2009. The CCN aims to build a pure content-oriented network, which means that the entire networking architecture and all the networking activities are based on the content themselves. Content discovery and delivery, networking packet routing are all based on content names. The CCN intergrades many features such as on-path caching, self content security, multicast, mobility, natively into its prototype, not as in IP that these features are realized by adding other overlay or services. The entire content name based networking design can make the content delivery more efficient.

The novel CCN proposal has many benefits but it brings also a big challenge for current hardware technologies, for example the memory requirement. On one hand, the transition from IP addresses to content names requires more memory space for storing the relative longer and more complex content names and also for more packets information. On the other hand, since in CCN the information exchanges are based on content names, but not on the end-to-end conversation, more and more networking packets will be generated. That can lead a more intense networking traffic which requires a higher routing performance. In another word, the memories which are implemented into a CCN router should provide a fast access performance. However today's fast memory chip can not provide a big storage volume or the large memory chips have a relative slow access performance. Then with the conventional implementation technique, there will be a tradeoff between the storage space and the performance speed for bring the CCN into real. Thus how to reduce the memory consumption is an important issue for the deployment of CCN solution in current networks. And furthermore in a CCN node, different components have different memory utilization structures. So the solutions of how to reduce the memory requirement for different components have to be different too. In this thesis I focused on how to improve the performances of different CCN node components and how to finally guarantee the performance of the entire node.

I firstly focused on the PIT element. In order to reduce the memory requirement and further improve the routing performance, I proposed a distributed PIT system which is based on the Bloom filter structure. The Bloom filter based system can largely reduce the memory consumption of the PIT table. The distributed design firstly can resolve the information retrieval problem which is a native shortcoming of Bloom filter. Secondly, the distributed system can make each CCN face manage its own PIT table, instead of using a global table read/write lock for a centralized table. Thus the treatment and the routing speed are improved. At last a two-level verification design can guarantee the false positive ratio is limited.

Thereafter I concentrated on the FIB element. In the origin CCN design, the FIB is filled by the flooded content advertisements. Considering of the huge number of potential content numbers, this method will explode the FIB table and also introduce a big volume of networking traffic. Facing this point, I proposed a content-aware dynamic CCN forwarding system which includes a content advertisement publish protocol, a FIB filling algorithm and a downstream forwarding element. The content publish protocol requires that each advertisement is forwarded only towards certain nodes, not flooded the entire networking. The FIB takes the consideration of the content popularity and memories only the information of contents that delivered towards it. And when a FIB forwards an *Interest* message which is unknown by the forwarding base, it sends the *Interest* packet to the "certain nodes" that receive the content advertisement. The downstream forwarding element is a table that is in charge of discovery the other potential content sources in order to take better advantages of the in-networking caches of the other nodes.

The third contribution of this thesis is about the interconnection issue of the CCN networking design and the CDN service. The CDN service brings already many advantages in IP network and it should also be supported in any other next generation networking solutions. However since the CCN does not support the negative response and there is no service such as DNS redirection or the address-based HTTP redirection, the interconnection between a conventional CCN node and a CDN repository node becomes an issue because the key point of interconnecting with CDN service is how to resolve the CDN repository miss-hit problem. In order to overcome this problem, I proposed in this thesis a system, which can be implemented into a classic CCN node, that can support the interconnection of the CCN networking structure with the CDN services. This system includes two elements. One element is in charge of sending the related *Interest* packets towards the CDN repositories. The other one is for discovering the CDN content miss case and recover this lost. With this interface, the conventional CDN nodes can be deployed into a CCN networking and this "upgraded" CCN node can also be connected with other classic CCN nodes in the traditional way.

To summarize up, this thesis contains three contributions that focus individually on different CCN node component in order improve the performance of one CCN node. Therefore they can be combined together and create an entire CCN node that is more efficient.

Keywords: CCN, performance, efficiency, memory,implementation, PIT, Bloom filter, forwarding, FIB, dynamic, content-aware, algorithm, protocol, CDN.

Contents

List of Tables

List of Figures

Résumé

L'évolution d'Internet IP et les limitations

Internet a été créé dans les années 70s pour échanger des informations ou communiquer. Les utilisations d'Internet étaient relativement simples, comme rechercher des informations sur des sites web, parler avec des amis grâce à des logiciels de messagerie instantanée, envoyer/recevoir des mails, télécharger des fichiers depuis des serveurs FTP, *etc.* Pour ces usages, les communications dans les réseaux sont réalisées selon un modèle de bout-en-bout, en établissant des tunnels de communications, depuis un point de terminal (un utilisateur par exemple) vers un autre point (Figure 1). Ce modèle, en vigueur pour les protocoles TCP/IP, était parfaitement adapté à ces usages.

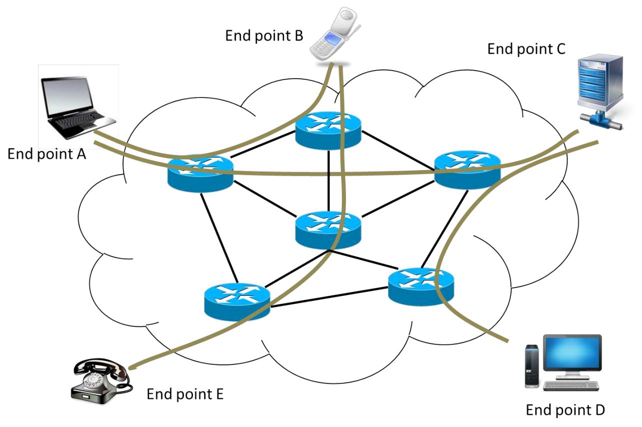


Figure 1: L'architecture de réseaux IP

Mais au fur et à mesure, Internet est devenu de plus en plus populaire et de plus en plus utilisé, ce qui a conduit à une évolution d'Internet et des technologies, les réseaux étant plus complexe qu'avant. Les terminaux ne sont plus simplement des ordinateurs bureautiques ou portables, mais sont maintenant des smart phones, des tablettes, des terminaux de jeux vidéo, *etc.* Les infrastructures réseaux ne se limitent plus à xDSL ou à la fibre optique; les réseaux WiFi, mobile 3G/4G, WSN et satellites sont maintenant largement déployés. La structure de TCP/IP et la couche de transport devient de plus en plus complexe pour gérer cette pluralité d'environnement. Plusieurs couches et services ont été ajoutés afin de s'adapter à ces besoins. Par exemple le mécanisme NAT [Tsi00] est appliqué pour résoudre le problème de manque d'adresse ; La couche sécurité [KA98] est ajoutée pour protéger les données des utilisateurs; Le service DNS [E^+11] est déployé pour traduire des adresses URL aux adresses IP. De plus, les fonctionnalités comme le multicast, la mobilité, le multi-homing, la communication sans fil, le SDN, le cloud computing *etc.*, sont ajoutées dans la couche transport du modèle TCP/IP, par l'intermédiaire de couches supplémentaires ou de patches protocolaires. L'effet le plus important de cette évolution est le changement des usages Internet. Au lieu d'avoir l'objectif de se connecter à des serveurs identifiés pour rechercher une information, les utilisateurs souhaitent maintenant récupérer les informations, sans se soucier de l'entité qui leur fournira. Selon des études, en 2013 59,5% de trafic réseaux étaient composés des vidéos en ligne (par exemple Youtube, Dailymotion, Netflix ou des services VoD d'opérateurs). Et ce pourcentage pourrait atteindre 69% en 2017 [Cis]. Des services tels que Twitter, Facebook attirent beaucoup de clients car les gens aiment bien de partager des informations personnelles avec des amis. Flickr, Instagram deviennent aussi très populaires. Çeci illustre que les utilisateurs n'ont pas uniquement des textes (profils, status) à partager. Si on analyse bien tous les phénomènes, on peut trouver que ce qui a changé est le concept fondamental des usages Internet. Ce qui intéresse les clients n'est plus l'endroit où il peut trouver (localisations) les informations, mais plus précisement les informations (contenus) elles-mêmes. Néanmoins, le modèle original d'Internet de type bout-en-bout n'est pas efficace pour de tels services de distributions de contenus. C'est pour cette raison que des applications P2P [SF02] et CDN [VP03] sont appliqués dans les réseaux.

Les motivations d'Information-Centric Networking

Pour remédier aux inconvénients mentionnés ci-dessus, des chercheurs ont réfléchi à une nouvelle solution d'Internet, mieux adaptée aux usages Internet actuels. Ces réflexions ont abouti à un nouveau paradigme réseau, dénommé Information-Centric Networking (en anglais) — ICN [MTP^+11]. Les réseaux ICN proposent de changer l'Internet, qui est actuellement basé sur les localisations des serveurs avec des adresses bien définies, vers une architecture basés sur le nom des objets, avec les fonctionnalités mentionnées précédement nativement intégrées (Figure 2). Dans un réseau ICN, les éléments de base du réseau sont les contenus, identifiés avec un nommage précis, plutôt que les machines (*e.g.* des serveurs, routeurs) identifiés avec leurs adresse IP. Toutes les fonctionnalités des réseaux comme le routage, la sécurité, sont aussi basées sur les noms des contenus. Comme dit dans le paragraphe précédent, avec IP, tous les échanges d'information sont basés sur des communications. C'est-à-dire qu'avant de commencer des activités, les clients doivent signaler au réseau avec quel point exact ils veulent communiquer. Ensuite le réseau a pour objectif de trouver ce point et d'établir une connexion entre les deux entités. Le réseau ICN est un modèle basé sur le récepteur (receiver-driven en anglais). C'est-à-dire qu'un utilisateur exprime seulement son intérêt sur des contenus au réseau. Ensuite c'est le réseau qui a pour charge de trouver les bons contenus et les meilleures sources pour ces contenus, en se basant sur leurs noms. Quand l'intérêt du client arrive finalement à une source de contenu, le contenu est délivré en suivant le chemin inverse du message d'intérêt jusqu'au client. Au final, le client est satisfait car il reçoit le contenu désiré, même s'il n'a pas connaissance de l'entité qui lui a fourni ce contenu.

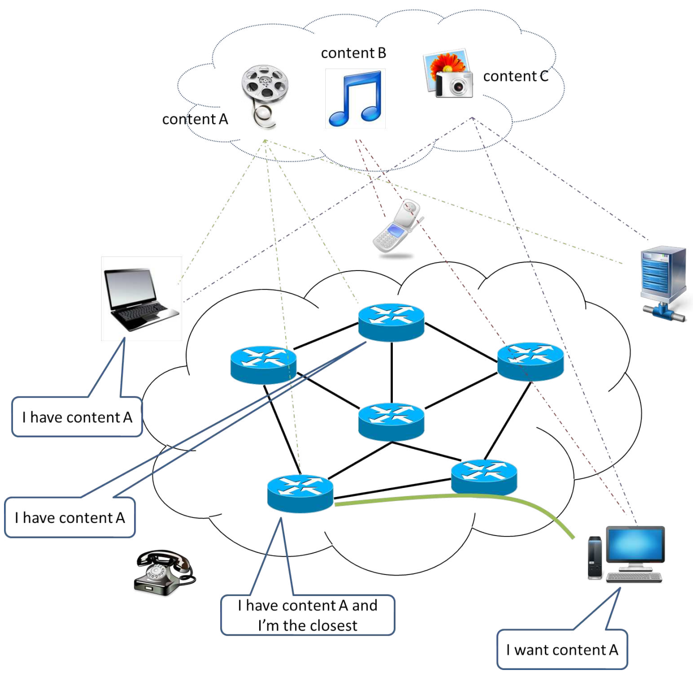


Figure 2: L'architecture des réseaux ICN

Outre le modèle qui est différent, par rapport à IP, ICN intègre aussi des fonctionnalités directement dans le réseau. Par exemple, ICN inclut nativement des avantages comme un nommage indépendant de la localisation, un routage basé sur les noms, la faculté de cacher des contenus dans les réseaux, le multicast, la sécurisation des contenus nativement intégrée, *etc.* Grâce à ces avantages, les réseaux ICN sont plus efficaces pour délivrer des contenus aux utilisateurs avec une meilleure qualité et permettent aussi d'améliorer la gestion des capacités réseaux des fournisseurs des réseaux. Voici les principales fonctions nativement intégrées dans ICN :

- **Le multicast:** En cas de multiples demandes d'utilisateurs différents pour un même contenu, seule la première requête est envoyée vers une source potentielle, les autres sont enregistrées et mis en attente dans le nœud ICN. Lorsque les données reviennent depuis la source de contenu, elles seront retransmises à chaque demandeur par l'interface sur laquelle la demande a été reçue. Ainsi, ICN peut nativement effectuer du multicast afin d'optimiser la livraison de contenu.

- **La mobilité:** Étant donné qu'ICN fonctionne sur un modèle non-connecté (il n'y a pas de connexion établie dans un réseau ICN), la mobilité des utilisateurs ne modifie pas le comportement des réseaux ICN. Leurs demandes, issues de différents endroits à différents moments, sont traitées indépendamment par les réseaux ICN, chacune comme une requête unique.

- **Le multipath:** Un objet de contenu peut être disponible dans plusieurs sources de contenu. Ainsi, chaque demande à un nœud ICN peut être transmise vers plusieurs destinations si plusieurs interfaces (par exemple, 3G, WiFi) sont disponibles sur le nœud ICN. Cela permet de partager la charge de trafic et de trouver les meilleures sources de contenu.

- **La sécurité:** ICN fournit une garantie de sécurité auto-protégée via des contenus cryptés et auto-certifiés, et non pas via des connexions de communication sécurisées comme IP. Seuls les utilisateurs autorisés peuvent déchiffrer les contenus.

- **Les caches dans réseaux:** Une caractéristique importante des réseaux ICN est la capacité de mise en cache de contenus directement dans les noeuds du réseau. Chaque morceau d'un contenu peut être mis en cache dans les nœuds de réseau se trouvant sur le chemin de la livraison du contenu, de sorte que les demandes ultérieures pourront être satisfaites plus rapidement, directement par les caches des noeuds ICN. Les paquets perdus pourront aussi être aussi récupérés plus rapidement par des retransmissions directes depuis les caches les plus proches.

Bien qu'ICN soit maintenant un sujet de recherche étudié par plusieurs équipes de recherche, il en est encore à ses premières phases. De nombreuses perspectives de recherche sont ouvertes. Dans les dernières années, plusieurs projets ou solutions ont été mises en œuvre par des équipes de recherche différentes, avec l'objectif de réaliser et déployer un réseau ICN; chacun de ces projets ayant des avantages et des intérêts différents. Parmi ceux-ci, nous pouvons citer: CCN [JST$^+$09] DONA [KCC$^+$07], NetInf [MA10], PSIRP [ZGR$^+$10].

Dans cette thèse, j'ai choisi CCN comme une cible de recherche. En effet, CCN est une solution ICN qui a pour but ultime de proposer un réseau complet basé sur des contenus et de finalement remplacer IP (contrairement à d'autres solutions qui proposent plutôt une approche overlays, toujours basée sur IP). De plus, il existe un prototype open source assez avancé, avec lequel nous pouvons faire des expérimentations et proposer des améliorations.

Content-Centric Networking

La solution Content-Centric Networking [JST$^+$09] (CCN) a été proposée par Van Jacobson et son équipe de PARC en 2009. Dès le début, elle a attiré beaucoup d'attention et il y a des nombreux projets qui travaillent sur cette proposition, par exemple le projet Named Data Networking (NDN) aux états unis [ZEB$^+$10]; le projet ANR CONNECT [Con] en France, *etc.* Dans des réseaux CCN, les localisateurs de réseaux comme l'adresse IP sont abandonnés. L'unité de réseaux dans CCN est un segment (aussi appelé *chunks* dans CCN) de contenu et chaque segment de contenu a un propre nom. Toutes les activités sont basées sur ces noms de contenu. Le concept fondamental de CCN est que lorsqu'un client veut un contenu, il envoie une requête d'intérêt (le paquet *Interest* dans CCN) qui contient le nom de ce contenu souhaité. N'importe quelle source de contenu qui a ce contenu peut répondre (par le paquet *Data*), avec le nom du message *Data* qui correspond au nom du message *Interest.*

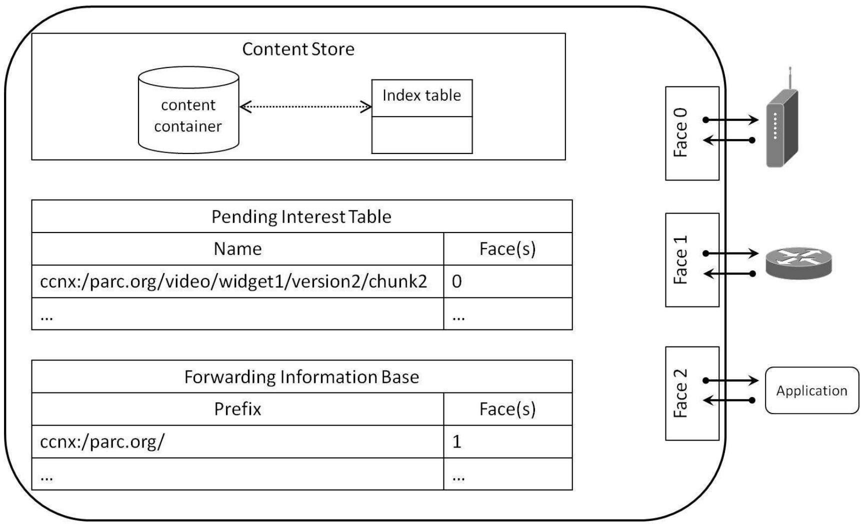


Figure 3: La structure d'un nœud CCN

Le nommage de contenus est au cœur de la solution CCN. CCN utilise une structure hiérarchique pour nommer des contenus. Un nom hiérarchique est organisé avec un préfix-suffixe, par exemple *ccnx:/parc.org/video/widget1/version2/chunk2*. Dans cet exemple, tous les contenus proposés par *parc* peuvent partager le même préfixe *ccnx:/parc.org/*. Un même contenu peut avoir plusieurs versions différentes et un contenu peut être divisé en plusieurs petits segments, afin d'adapter la couche de transport. Ainsi, chaque nom de CCN se termine par la version et les informations de segments qui peuvent simplifier la découverte de contenu.

Un nœud CCN est principalement composé de trois éléments (Figure 3): la *Pending Interest Table (PIT)*, la *Forwarding Information Base (FIB)* et le *Content Store (CS)*. CCN utilise un concept de *face* dans son architecture. La face est similaire à la notion d'interface en IP, mais elle inclue aussi des interfaces vers des applications, et pas seulement les connexions physiques.

- **La table PIT** a deux fonctionnalités principales. La première est qu'elle mémorise temporairement des messages *Interest* que le noeud reçoit avant de les transmettre ensuite au noeud suivant. Grâce à cette table, en retour des données, le paquet *Data* peut suivre les chemins inverses et finalement arriver jusqu'aux clients demandeurs. Le deuxième rôle de PIT est d'éviter de multiples envoi des mêmes messages *Interest*. Lorsque plusieurs messages *Interest* qui demandent un même contenu arrivent sur un noeud, seul le premier est renvoyé pour chercher le contenu; les autres restent dans ce nœud et attentent la réception du contenu. Une fois reçues, les données seront retournées sur chacune des faces présentes dans la PIT.

- **La table FIB** de CCN est similaire à celle d'IP. Elle est utilisée pour gérer les informations de transfert des paquets *Interest* vers des sources qui ont les contenus demandés. La table FIB est remplie par des publications de contenus qui sont publiés par des fournisseurs de contenu.

- **Le Content Store** est un cache (ou une mémoire tampon) installé dans les nœuds CCN. Lorsqu'un nœud reçoit des données (message *Data*), selon les stratégies de caches définies, le nœud peut sauvegarder une copie de ces données dans son CS pour répondre aux demandes ultérieures.

Pour obtenir un contenu dans un réseau CCN, une entité cliente transmet dans le réseau un message *Interest* relatif à ce contenu et comprenant le nom du contenu. Sur réception de ce message *Interest* sur une face réseau donnée, un nœud CCN, vérifie s'il dispose du contenu recherché dans son Content Store. S'il dispose du contenu recherché, il le transmet par l'intermédiaire de la face sur laquelle la message *Interest* a été reçu, à destination de l'entité cliente, éventuellement par l'intermédiaire d'autres nœuds. S'il ne dispose pas du contenu recherché, il vérifie dans sa table PIT s'il a déjà reçu un message *Interest* relatif au même contenu via la même face ou une autre face. Si ce n'est pas le cas, il mémorise dans sa PIT le nom du contenu dans le paquet *Interest* en association avec un identifiant de la face par laquelle ce message *Interest* a été reçu. Il envoie ensuite ce message *Interest* dans le réseau selon la table FIB. Si, par contre, sa table PIT comprend une entrée qui a déjà le nom du contenu recherché en association avec l'identifiant d'une autre face, il ne transmet pas la requête reçue, mais met à jour cette entrée associée de la PIT avec l'identifiant de la face par laquelle ce paquet *Interest* a été reçu. En effet, si la PIT comprend déjà le nom du contenu recherché, cela signifie qu'un message *Interest* relatif à ce contenu a été précédemment reçu et transféré dans le réseau, et que le nœud est en attente d'une réponse à ce message *Interest*.

Lorsqu'un message *Data* est reçu sur une face, le nœud consulte sa table PIT pour connaître les faces ayant demandées ce contenus et transmet le paquet *Data* sur toutes les faces qui sont associées au nom du contenu. La PIT permet donc de retrouver le chemin que doit emprunter ce message *Data* pour atteindre le ou les clients qui l'ont demandé. Après transmission de ce message *Data*, l'entrée associée dans la PIT sera supprimée.

Contributions

L'objectif global de ma thèse est de garantir et améliorer les performances des réseaux CCN. Plus concrètement je me suis concentré sur les performances des nœuds, avec à l'esprit l'aspect de limitation des hardwares. En effet, CCN peut effectivement apporter des avantages indéniables, mais le matériel hardware actuel ne permet pas de supporter une solution CCN. Par exemple d'après la structure originale de CCN, un nœud a besoin de mémoire pour mémoriser les noms de contenus. Mais les mémoires offrant de grands volumes de stockage ne supportent pas des temps d'accès rapides. Donc les performances de routage sont très réduites. Ceci a motivé mes travaux de thèse et conduit à la réalisation de trois contributions majeures, basées sur les trois différents composants du nœud CCN.

La distribution de la table CCN PIT en utilisant des filtres de Bloom
Une PIT classique de CCN est réalisée comme une table centralisée basée sur des tables de hachage. On constate que les noms de contenu dans CCN sont de grande

taille, et très variable. Un nœud CCN, pour traiter un message *Interest*, doit donc stocker et manipuler les noms de contenu représentés sur un nombre quelconque de bits, ainsi que le ou les identifiant(s) de face. La table PIT qui mémorise tous les paquets d'*Interest* reçus, ainsi que l'information relative au chemin que le contenu doit emprunter peut alors constituer une faiblesse pour un réseau CCN du fait de la taille que cette table peut atteindre. En effet, les réseaux CCN étant destinés à acheminer tous les contenus mis à disposition sur le réseau Internet, on comprend qu'une telle table peut mémoriser des méga-, voire des giga-octets d'informations. On sait par ailleurs que plus la taille de la table PIT est importante et plus les temps d'accès aux données qu'elle contient sont importants. Cela peut nuire considérablement aux performances d'un réseau CCN.
Dans cette contribution j'ai proposé une nouvelle structure de table PIT de type distribué. Dans ce système, à chaque face d'un nœud est associée une PIT distribuée (appellée PITi), propre aux messages *Interest* reçues sur cette face. Par rapport à la mise en œuvre connue d'un nœud CCN qui comprend une table PIT globale, chaque PITi dans un nœud est de taille beaucoup plus petite que la table globale. En effet, la PITi dédiée à une face ne mémorise que les messages *Interest* reçues sur cette face. Par ailleurs, la table étant dédiée à la face il n'est pas nécessaire de mémoriser le ou les identifiant(s) de face sur laquelle/lesquelles les paquets *Interest* ont été reçues. Les PITis étant plus petites, un accès à ces tables est plus rapide, ce qui optimise le temps de traitement routage. Ce gain au niveau de chaque nœud d'un réseau contribue à améliorer les performances du réseau.

Comme il n'est plus nécessaire de mémoriser les identifiants de face dans les PITs, j'ai aussi proposé que les PITis soient mises en œuvre au moyen de filtres de Bloom avec compteurs. Un filtre de Bloom, avec ou sans compteurs, est une structure de données probabiliste compacte dont la taille est fixe et indépendante du nombre d'éléments contenus. Une telle table peut donc être extrêmement compacte. Une telle structure est très intéressante pour mémoriser une très grande quantité de représentations de noms de contenus de taille variable. Le filtre de Bloom avec compteur est adapté à l'enregistrement d'éléments, dans le cas présent les représentations des noms de contenus après application de fonctions de hachage à ces noms de contenus, et à la suppression d'éléments. Le filtre de Bloom est donc tout à fait adapté pour la mémorisation de représentations de noms de contenus. Une fois qu'un nom de contenu est mémorisé dans une PITi mise en œuvre par un filtre de Bloom, le nœud a besoin d'interroger le filtre pour savoir si le nom de contenu est présent dans le filtre, et de supprimer ce nom de contenu, lorsqu'un message *Data* a été traité. Ainsi, les filtres de Bloom avec compteurs optimisent la taille mémoire requise par la table PIT et contribue à optimiser les performances lors de traitement dans le réseau.

Cependant, les filtres de Bloom sont sujets aux faux-positifs. En effet, bien que l'on sache avec certitude qu'un élément est absent du filtre de Bloom, on ne sait qu'avec une certaine probabilité qu'un élément peut être présent dans le filtre. Afin de réduire la probalilité de faux-positifs, une seconde vérification avec un filtre de Bloom binaire global (appellé Shared Bloom Filter – SBF) est ajoutée, après la vérification réalisée dans une PITi sous condition que le résultat de la vérification de PITi soit positif. A cet effet, lorsqu'un message *Interest* de contenu arrive sur une face, la PITi associée à la face est interrogée sur la présence du nom du contenu requis. Si la PITi indique

que le nom du contenu est présent, alors l'interrogation du SBF permet de vérifier si l'on est en présence d'un faux-positif. En effet, si le SBF interrogé sur la présence du nom du contenu indique que le nom du contenu est absent, alors cela signifie qu'on est en présence d'un faux-positif puisque cela indique qu'aucun message *Interest* relatif à ce nom de contenu n'a été acheminé. Par ailleurs, si le SBF indique que le nom du contenu est présent alors on considère que c'est un faux-positif et que ce paquet *Interest* est une requête dupliquée. Ce paquet *Interest* est donc bloqué. Les résultats de simulations montent que la taille nécessaire pour implémenter la table PIT est largement réduite avec ma solution. Il est donc possible de réaliser cette table PIT avec les mémoires rapides d'aujourd'hui. La probabilité et l'impact de faux-positifs sont significativement réduits grâce au second filtre de Bloom global.

Une table CCN FIB dynamique basée sur les contenus Dans CCN, les routages de messages *Interest* sont réalisés à partir des noms de contenu et les informations de routage sont maintenues dans la FIB. La FIB est remplie par les publications de contenus propagés dans les réseaux. Cela signifie que si un fournisseur de contenu souhaite publier un contenu, il faut propager une publication dans le réseau afin que les autres nœuds sachent que ce contenu est disponible. Chaque nœud du CCN qui reçoit cette publication va ajouter le nom du contenu ou le préfixe dans sa FIB, avec la face sortante qui est calculée sur une théorie des graphes de routage et l'identifiant de l'annonceur (protocole OSPFN). Mais j'ai trouvé que cette méthode n'est pas dans la philosophie de CCN, car elle est toujours basée sur les identifiants des fournisseurs ou des autres nœuds de contenu. La performance est également limitée car, comme toutes les publications sont propagées dans les réseaux, les publications atteignent tous les nœuds disponibles et les FIBs de ces nœuds CCN devraient donc contenir toutes les informations de routage qui couvrent tous les contenus disponibles sur l'ensemble du réseau. La taille de la FIB serait donc immense. De plus, avec cette propagation globale, chaque nœud de réseau a la même vue des contenus disponibles dans les réseaux. Ce n'est pas efficace parce que, en réalité, les différentes régions ont des popularités de contenu différentes. À mon avis, le système basé sur les contenus doit également intégrer ces comportements de contenu comme la popularité.
Sur la base de ces analyses, j'ai proposé un système de transfert CCN dynamique basé sur les contenus transitant par le nœud. Dans cette proposition, la FIB est remplie directement par les paquets *Data* entrants, et non par les annonces de publications. Lorsqu'un nœud reçoit un message *Data* en réponse à un message *Interest*, il va mémoriser le préfixe ou le nom de domaine ainsi que la face entrante du message *Data* dans sa FIB. Ainsi lorsqu'un nœud a un message *Interest* à envoyer, s'il y a déjà des informations associées à ce préfixe, ce message *Interest* sera envoyé selon ces informations. Par contre, si aucune information n'est définie, une *(*face défaut), qui permet le routage vers certains nœuds pré-définis, par exemple des routeurs de bordures d'un AS, est utilisée. Dans ce cas, les messages *Interest* sont redirigées vers les *(*faces défaut). Ce système prend également en considération des sources de contenu (ils peuvent être des Content Stores dans les routeurs, des caches dans les set-top box, ou des terminaux de P2P, *etc.*), qui sont au niveau du nœud en aval. En effet, étant donné que les segments de paquet *Data* sont mis en cache dans les Content Stores sur le chemin de la livraison, ces nœuds peuvent également être considérées comme

fournisseur de contenu potentiel. Cependant, la conception originale de transfert de CCN origine ne considère pas cela parce que les Content Stores n'annoncent pas les contenus qu'ils cachent. J'ai donc proposé d'ajouter une table qui s'appelle DIFT (Dynamic Interest Forwarding Table). La DIFT a des relations avec la PIT. En effet, les informations sauvegardées dans PIT sont utilisés pour livrer des messages *Data*. Elles peuvent aussi être considérées comme des informations de transfert qui nous indiquent les Content Stores dont les nœuds viennent de recevoir des messages *Data*.

Interconnexion des réseaux CCN avec des services CDN La technologie CDN est déjà largement déployée dans le réseau IP. Elle apporte de nombreux avantages. Du point de technique, il peut réduire la latence de livraison de contenus, équilibrer la charge dans les réseaux, et offrir aux utilisateurs une meilleure qualité d'expérience. Du point de vue de l'entreprise, la mise en œuvre de serveurs CDN permet d'économiser le trafic de bordure entre les fournisseurs de réseaux différents. Certains chercheurs pensent que les caches sur le chemin, comme le Content Store de CCN, peuvent remplacer les services CDN. Personnellement, je pense que toute proposition de réseau du future devrait prendre en considération le service CDN, y compris le CCN. En effet, si un nœud de CCN sur le chemin veut effectuer les mêmes fonctionnalités qu'un serveur CDN, il doit être équipé d'un grand volume de cache de l'ordre du Tera-octets. Ce genre de routeur hybride CCN / CDN ne peut donc pas offrir une bonne performance, car il doit fonctionner très rapidement à la fois pour le routage et pour la gestion de contenu CDN, ce qui est critique pour les technologies de mémoire d'aujourd'hui. A mon avis, le service CDN doit être intégré de manière indépendante.

Cependant, certaines fonctionnalités du prototype CCN limitent le déploiement de CDN directement dans un réseau CCN, en raison du problème *miss-hit* CDN. En effet, d'une part, CCN, de par son protocole, ne supporte pas de réponse négative. Deuxièmement, il n'y a pas de notion de localisation dans CCN, donc la redirection n'est pas possible pour les serveurs CDN.
Face à ces inconvénients, je propose une nouvelle architecture de nœud CCN (nommé cRouter) et un algorithme de traitements des paquets, qui permet l'interfonctionnement des serveurs CDN et des réseaux CCN, sans modifier le prototype CCN origine. Dans ce système, les informations de transfert vers des serveurs CDN ne sont pas indiquées dans la table FIB, mais dans une nouvelle table, appellée RFT (Repository Forwarding Table). La RFT contient des noms de domaines ou des préfixes de noms, dont les contenus peuvent être présents dans les serveurs CDN (accord entre fournisseurs de CDN et fournisseurs de contenus). Cette table peut être configurée de manière manuelle ou dynamique avec des protocoles de propagation. La FIB elle mémorise uniquement les informations associées aux serveurs d'origine. Ceci est nécessaire car les serveurs CDN ne contiennent qu'une partie des contenus des fournisseurs (les plus populaires souvent) et il y a donc risque de ne pas trouver le contenu si le paquet *Interest* est transmis uniquement au nœud CDN (ce que nous appelons *miss-hit*). Afin de détecter ce CDN miss, j'ajoute une table, qui s'appelle PRIT (Pending Repository Interest Table), pour mémoriser temporellement les noms et les faces entrantes des paquets *Interest* qui sont rédigés aux serveurs CDN. Ce mécanisme est lié à la retransmission du côté client. Lorsqu'un cRouter reçoit un message *Interest*, il va tout

d'abord vérifier s'il est présent dans le RFT. Si le RFT ne trouve pas d'information relative à ce message *Interest*, ce message *Interest* sera traité normalement par PIT et FIB. Si l'information existe , un processus de vérification dans PRIT est activé. Si dans PRIT, il n'y a pas d'entrée associée au nom de ce message *Interest*, ou s'il y a une entrée mais la face entrante de ce message *Interest* n'apparaît pas dans cette entrée, ce paquet *Interest* sera envoyé vers des serveurs CDN en fonction de la table RFT. En même temps, le nom et la face entrante de le message *Interest* sont sauvegardés, ou mis à jour dans la table PRIT. Si ce n'est pas le cas, c'est-à-dire que la PRIT trouve bien une entrée qui contient déjà ce nom et cette face de paquet *Interest*, ou une liste de face vide, il considère ce message *Interest* comme une répétition de message *Interest* et que le serveur CDN n'a répondu pour cette demande. Dans ce cas, ce message *Interest* sera envoyé au serveur d'origine avec le processus normale de PIT/FIB. L'entrée dans PIT est créée avec la face entrante de ce message *Interest*, mais aussi avec toutes les faces qui sont listées dans l'entrée trouvée dans PRIT. Parallèlement, l'entrée dans la PRIT est conservée pour un moment, avec la liste de face vide, pour les prochaines vérifications. Lorsque le message *Data* est reçu, il sera envoyé aux clients soit selon la table PRIT s'il vient via une face connectée à un serveur CDN, soit selon la table PIT s'il vient via une face connecté à un nœud autre qu'un serveur CDN.

Les avantages de cette solution sont que les fonctionnalités des différentes entités de réseaux sont bien séparées. Le fournisseur de services CDN peut se concentrer sur la gestion et la diffusion des contenus et les opérateurs peuvent se focaliser sur la façon d'améliorer les performances réseau. De même, les constructeurs peuvent aussi se concentrer soit sur des serveurs multimédias efficaces, soit sur des routeurs plus performants, selon leur activité.

Mots clés: CCN, performance, efficacité, mémoire,implémentation, PIT, filtre de Bloom, forwarding, FIB, dynamique, content-aware, algorithme, protocole, CDN.

Chapter 1

Introduction

I would like to start this thesis by introducing the context of today's Internet. In this chapter my goal is to provide a brief presentation of the IP networking evolution and the motivation for the ICN/CCN paradigm. Then I make a short introduction of the objectives and the contributions of this thesis. At last this chapter ends by an outline of the entire thesis.

1.1 The evolution of IP network and the motivation for ICN

1.1.1 Limitation of IP

Since the current Internet was designed in 60s-70s, it has played a more and more important role in people's life. From the beginning, the Internet is designed as an end-to-end model and runs on top of five layers of the TCP/IP protocol stack with the intention to connect a few machines.

1. **Physical layer**. The physical layer includes all the medium for the transmission of the signals, for example the optical fiber, the twisted pair cable, or the air (WiFi).

2. **Data link layer**. The data link layer provides a reliable link between two directly connected devices. It includes the functions as error correction, the packet retransmission control, loop detection, and channel assignment, *etc.*

3. **Internet layer**. There might be multiple data links between two devices. The role of the Internet layer is to choose the optimal routes and switch points between the equipment. The TCP and IP protocols are included in this layer.

4. **Transport layer**. In the Internet layer, packets are delivered in the hop-by-hop communication model. The role of Transport layer is to establish a virtualized end-to-end link between two remote devices that are communicating.

5. **Application layer**. The Application layer provides all the Internet application services or protocols, for example the HTTP, FTP, BGP or DNS *etc.*

Based on this Internet paradigm, all the Internet information exchanges are realized by establishing the communication channels between the networking equipment (as shown in Figure 1.1). This host-centric Internet design perfectly matched the simple Internet usage at the early age, since early Internet applications or protocols were also end-to-end communications, for example web surfing on a certain web site, instance message chat with a close friend, sending e-mails, or FTP download from a known server, *etc.*

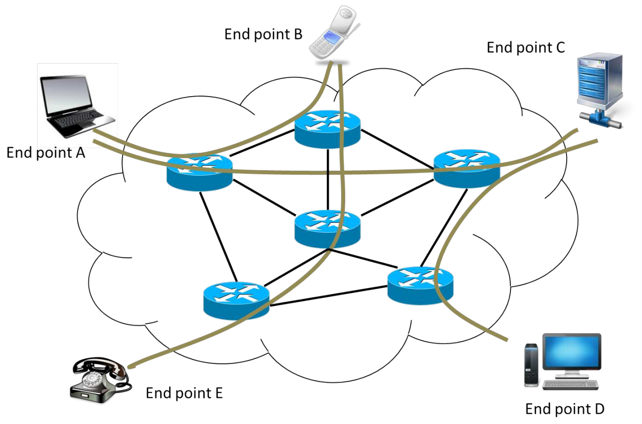


Figure 1.1: IP networking architecture

The science and technology never stop from evolving. In the last decade, the Internet has experienced an explosive revolution. The networking link layer is no longer limited at the optical fiber or xDSL technique. Various technologies are deployed for Internet information exchange, from WiFi, mobile 3G and 4G, to ZigBee [BPC+07], Satellite, *etc.* The end-user terminals which support the Internet experience cover from the PC, laptop, smartphone, tablette, to the video game terminals, music players, set-top box, *etc.* NAT [Tsi00] is proposed because of the lack of IP addresses. DNS [E+11] is deployed because we need to translate the human readable URLs to the routable IP addresses. Information security becomes also important, for example E-commerce is popular today, thus IPsec layer [KA98] is also addressed. Not to mention, multicast, multi-homing [SX01, NB09], mobility, security for multi-homing [MPH+12], security for mobility, cloud computing, SDN, *etc.*, more and more patches or overlays are added on the current Internet protocol stack in order to meet the needs. More importantly, the Internet usage is changing as well. It is switching from the host-centric to a content-oriented model. On-line video (*e.g.* Youtube, Dailymotion, Netflix, VoD services (Video-on-Demand), *etc.*) becomes already the major contribution (59.5% in 2013, and 69% estimated for 2017) of the networking traffic [Cis]. Twitter, Facebook attract many users who enjoy sharing their daily life on-line with others. Flickr, Instagram gets also success because not only the text information is what people want to share, but also the multimedia information. The Internet becomes more and more complex and bottlenecks start to appear. For example:

- If one hundred million users are watching the World Cup football game on line at the same time, and even if each user has only a basic 100kbps downlink connection, it leads to a total throughput of 10Tbps, which is much more than the tier one throughput capacity of a country. In order to save the bandwidth, the live videos are currently retransmitted from CDN nodes [VP03] which are deployed by the video service providers. Sometimes the P2P [SF02] acceleration technology is also implemented. That is why when we watch a live football game online, we have a few minutes delay. This phenomenon is because that even if two football fans are neighbors, the server has to send two data packets to each of them. So what about a solution where the server only sends one copy and lets the routers located on the last mile send this packet twice to the end-users? It would be network-efficient, however the standard IP structure does not support it.

- Today more and more mobile devices are connected to the Internet and people move a lot. If one move from one WiFi hot-spot to another one, or switch from a WiFi connection to a cellular network (3G, 4G), the established Internet connection will be down, because the IP address is changed and the route has to be recalculated. That is why it is difficult to keep an efficient end-to-end connection in a mobile IP network.

1.1.2 Motivation of ICN

Facing these shortcomings, many research communities are motivated to develop a so called Information-Centric Networking paradigm (ICN) [MTP$^+$11]. Because if we look carefully at the deep common points of those problems, we can observe that all services rely on the end-to-end connection. But the current Internet usage shows that what people care about is no longer “where” it can get the information that they are interested in, but “what” the information actually are, which means the end-to-end model is no longer the obligation.

The ICN paradigm aims to shift Internet model from the current complex IP structure to a simple and generic one based on the information objects, with all the services that are mentioned above embedded (as shown in Figure 1.2). In ICN network, the basic networking unit is the named content objects, no longer the identified nodes (*e.g.* servers, routers, terminals, *etc.*). All the networking activities are based on the named content objectives. In IP, before every activity, the end-users should tell the network clearly which location they want to communicate with. The job of the network is to find out the exact location and establish a conversation channel. The ICN is a receiver-driven networking model, where end-users only express their interests for a given content, the entire network is in charge of routing the requests based only on the content names towards the best content containers and delivering the contents through the reverse paths to the end-users. The end-users in fact have no idea about where these copies of contents are from. The ICN aims to build the attributes directly into the networking design. It natively includes the features as location-independent naming, name-based routing, in-networking caching, native multicast, self-secured content, *etc.*

With this design, ICN can optimize and simplify the content delivery experience and leverage the service providers’ infrastructure capabilities, such as:

1. **The mobility:** Since the ICN communications are not based on the established connections, the movements of end-users will not affect the networking activities. Their requests from different locations at different times are processed independently by the network.

2. **The multicast:** The requests for a same content from different end-users will aggregate in one ICN node and only the first one is sent out. When the content is replied, each appended request will get a copy of the content. Thus the ICN can natively perform the multicast in order to optimize the content delivery.

3. **The multipath:** A given content object can be available in more than one content container. Thus each request at an ICN node can be sent out via multiple available interfaces (*e.g.* cable, WiFi) in order to share the traffic load and find out the best content containers.

4. **The security:** ICN provides a self-protected security guarantees via the encrypted contents and the self-certified content names, not via the secured communication channels. Only the authorized users can decrypt the content.

5. **The on-path caching:** The most import feature of ICN is that it provides the on-path caching capacity of the contents. Each piece of content object can be cached in the networking nodes along the delivery path, so that the subsequent requests can be replied more rapidly direct from the caches, and the lost packets can be also covered faster.

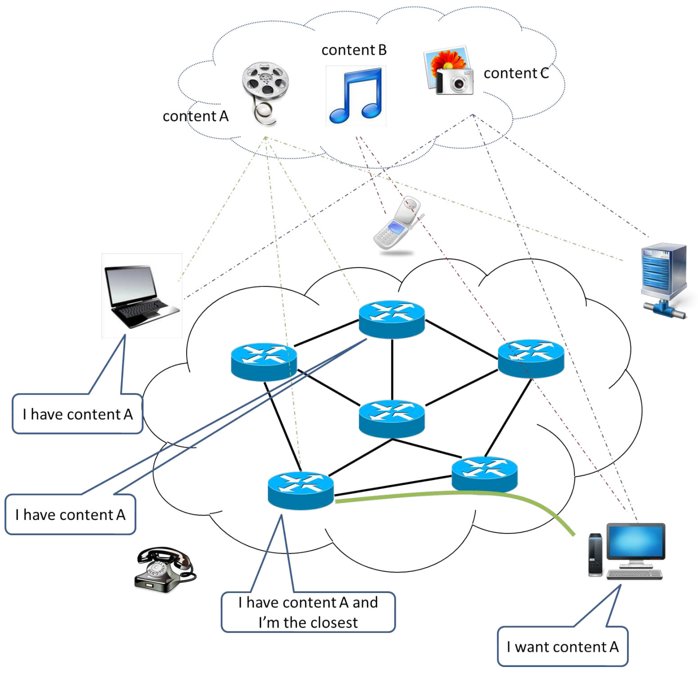


Figure 1.2: ICN networking architecture

Although ICN enters now into the main stream of networking research, it is still in the early stage. In the past few years, many projects have been carried on in order to propose a concrete ICN solution to deploy it in reality. Different projects use distinct notations to indicate their design choices and features. For example the CCN [JST+09], DONA [KCC+07], NetInf [MA10], PSIRP [ZGR+10],

In my thesis I chose the CCN as my research target, because it is the most advanced and mature proposal. Its ultimate goal is to replace the entire IP based networking structure with the named content based design, although the others are more about proposing a optimized overlay candidates.

1.2 Objectives and Contributions

The global objective of this thesis is to guarantee and improve the CCN networking performance.

In my research work I focused on how to improve the performance of a single CCN node in terms of overcoming the hardware limitations, packets processing, content-based forwarding and content discovery. The motivation of these choices is that, the CCN protocol and its services are still in the early stage. There is no existing mature service in business yet. Thus it is a little early to talk about the quality of a certain service. And on the other hand, the performance of each single node is the base of the performance of the connected network. Hence I think it is important to start by focusing on the single node performance.

This thesis mainly contains three contributions that are related with the three CCN node components:

- I proposed a new distributed CCN PIT design which is based on the Bloom filter structure. The original CCN PIT is designed as a hash table based structure. Hash table solutions, which have a $\mathcal{O}(1)$ time complexity, are largely used in IP applications. However they require too many memory space for CCN PIT usage. Since CCN content names are longer and more complex than IP addresses, a relative high packet arrival rate makes the memory requirement of implementing hash table base PIT exceed current fast memory volume limit. Thus the forwarding performance is limited. In order to reduce the memory space consumption, I proposed a Bloom filer based PIT solution in a distributed manner. The implementation of Bloom filter can significantly reduce the memory requirement, and the distributed structure can tackle the information retrieving problem of Bloom filter. In this contribution I carefully present the solution. And the evaluations validated the advantages of this solution. At last, I also made an example for giving some reality deployment recommendations.

- I considered the CCN forwarding information design. The CCN *Interest* messages are forwarded based on the content names. The routing information of *Interest* packets are maintained by the Forwarding Information Base (FIB) table. This table is similar as the FIB in IP network. It contains the prefixes of the content names (*a.k.a* ContentNames in CCN paradigm) and the outgoing interface (*a.k.a* face in CCN). However the FIB is filled by the broadcast content advertisements. That means if a content provider wants to publish a content, it should propagate a content advertisement into the network in order to announce the availability of this content. Each CCN node that receives this content advertisement will add the content name or prefix into its FIB, together with the associated outgoing face which is calculated based on a routing graph theory and the Id of the announcer. But I found this method is not CCN-friendly, since it is still based on the identifiers of the content providers. The performance is also limited. Because with this manner, first of all, the FIBs should contain all the routing information that allow to access to all the available contents in the entire network. The size of FIB will be huge. Second of all, in the original manner each networking node has the same view of the available content in the entire network. This is not efficient because in reality different regions have different content popularities. In my opinion the content-centric forwarding scheme should also consider the behaviours of contents, for example the content popularity. Based on these analysis, I proposed a content-aware CCN forwarding structure. In this proposal, the CCN FIB forwarding information is realized directly based on the incoming content packets, not on the broadcast content advertisement. This scheme takes also into consideration of the Content Stores that are at the downstream node. Since the content packets are cached in the Content Stores along the delivery path, these nodes can also be considered as potential content provider. However the original CCN forwarding design does not consider this because the Content Stores do not send the advertisements even they have the contents.

- In this contribution I studied about the deployment of CDN services in CCN network. The CDN technology has already been largely deployed into IP network. It brings many benefits. From a technical point, it can reduce the content delivery delay and offer the end-user a better Quality of Experience. From the business point, implementing CDN servers can economize the peering traffic between different IPSs. I think any next generation networking proposal should support the CDN service, including CCN. Indeed, some researchers argued that such on-path cache can replace the CDN services. However I doubt about this consideration. The reason is if a on-path CCN node wants to perform the same functionality as the CDN server, it should be equipped with a large space cache in the order of Tera Bytes. This kind of hybrid CCN/CDN node thus cannot offer a good performance, since it should perform fast both for packet routing and for CDN content management, which is critical for today's memory technologies. I believe that the CDN service should be integrated in an independent manner. Some features of CCN protocol restrict deploying the CDN directly in CCN network, because of the CDN miss-hit issue. Firstly, the CCN does not support the negative reply. Secondly, there is no locator notion in CCN, thus the direct redirection is not possible for CDN servers. Facing these inconvenient, I proposed a new CCN node architecture and packet process algorithm, which allows the interworking of CDN servers and the CCN core network, without modifying the original CCN prototype. The advantages of this solution is that the functionalities of different entities are well separated. The CDN service provider can concentrate on the content delivery management issue, and the operators can focus on how to improve their the networking performances.

1.3 The organization of dissertation

This thesis contains six chapters. The Chapter 1 and Chapter 2 present a broad context and background of this work. The Chapters 3, 4 and 5 detail the three main contributions of this thesis. The thesis will ended with a conclusion and a perspective future work in Chapter 6 and Chapter 7, respectively.

Chapter 1, (*i.e.* this current chapter) , presents the motivation of the ICN netwoek and why I choose CCN as the candidate. It introduces also the objectives and the brief description of each contributions.

Chapter 2 gives firstly a brief introduction of some other ICN solutions. After that I present the details of the CCN network, including the networking design, content based routing, naming issue and on-path caching feature. In this chapter I present also a summary of the current memory technologies, which is running through this thesis.

Chapter 3 presents a new distributed PIT architecture which is based on the Bloom filter. Compared to the hash table based solution, this proposal can largely reduce the memory space requirement, so that the deployment of CCN node into real become possible facing the current memory chip limit.

Chapter 4 presents a new dynamical CCN content-aware forwarding design. Based on the incoming content packets, this scheme contains three main contribution: a new content advertisement protocol, a new FIB filling protocol and a downstream *Interest* message forwarding protocol. In this chapter I will present the details of each component. I also give some evaluations to prove that this proposal is space efficient, and since it is automatically aware of the local content popularities, the performance is better.

Chapter 5 presents a new kind of CCN node — cRouter. cRouter can support the classic CDN server interconnect with the CCN network. In this chapter I will present the details of the cRouter, and explain how it works to interconnect the CDN service and CCN core network, in considering of the CDN miss and CDN forwarding issues.

Chapter 6 makes a global conclusion of this thesis. In my thesis the three contribution focus on different CCN element, but they are not independent from each other. In this chapter I will summarize that the global objective if this thesis is to propose a redesigned CCN node structure with high performance and that meet the current hardware technology limits.

Chapter 7 discuss some of the potential future works that can improve these three years research.

Chapter 2

Background

2.1 General Introduction

In Chapter 1, I have presented the motivation and some features of ICN. In this Chapter, I first introduce some ICN solutions, and then I give a more detailed description of the CCN design. Finally I briefly summarize the characteristics of current memory technologies, since it drived my work for a realistic CCN node.

2.2 Some ICN projects

Since the ICN paradigm has been raised, many research forces proposed their ICN solutions or projects, with different architectures. In this section, I present some of them that are relative advanced and mature.

2.2.1 NetInf and SAIL

Network of Information (NetInf) [MA10, WWWd] solution is a networking architecture, proposed by the European FP7 project named 4WARD and its follow-up project named SAIL.

NetInf networking structure is based on a Naming Resolution System (NRS) and a Multiple Distributed Hash Table (MDHT) routing mechanism. In NetInf, the concepts of content representation and the data object of a content are clearly distinct. An *Information Object (IO)* is for directly referring a piece of content object and a *Bit-level Object (BO)* is the content data itself. An IO contains three information: a globally unique identifier of the content, a set of meta-data and a *Data Object* (DO) which can be regarded as a reference of the payload of the actual content — the BO. The content identifier contains the type of the content (such as text document, audio file, image, web page or live streams) and the hash of the owner's (or publisher) public key. The meta-data field contains a *metalist* which provides semantic information about the content that can be helpful for the search service, like the bit rate and codec of an audio recording or the author and the abstract of a document.

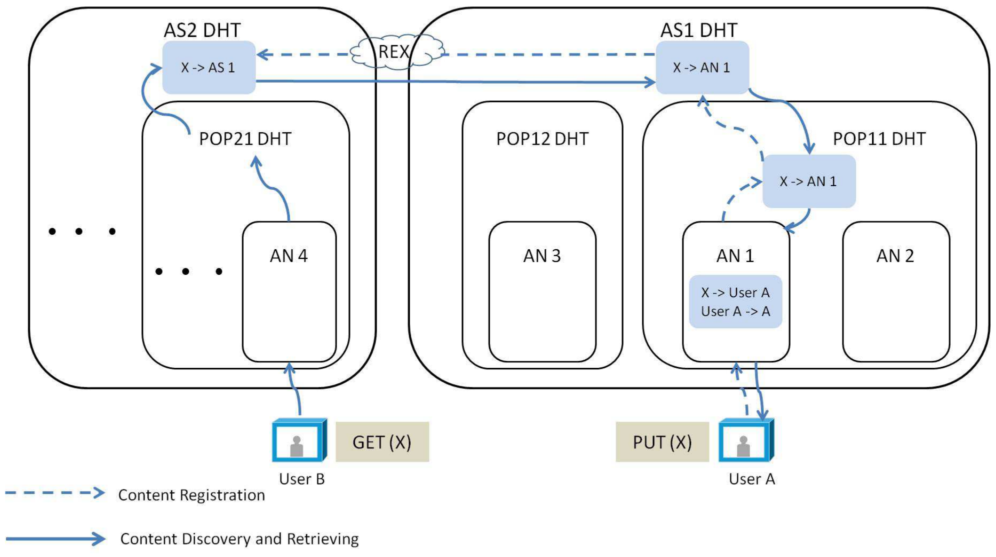


Figure 2.1: Routing scheme of NetInf

The roles of content authors and publishers in NetInf are different. The authors are the ones who create, sign and modify the IO. A same IO can have different versions and different versions can have different authors and different signatures. The publishers are not authorized to modify or sign the IOs, but they can redistribute the contents.

The NetInf Name Resolution System (NRS) takes a content identifier or a set of attributes which describe the searched object as the input, and returns a set of binding records for IOs that matches the input. The IOs include a reference (DOs) that directly or indirectly can be used to retrieve the BO. This means in NetInf system a two-step resolution is possible. The application or user can chose in the returned IOs list which one to select for requesting the corresponding BOs, based different criteria (cost, download speed, definition, quality, *etc.*).

Netinf defines different level name resolution zones which are realized by a Multiple DHT (MDHT). Each zone is responsible to persistently store a BO with corresponding identifying IO. When a client requests an object, a first DHT lookup is made at the first level (*e.g.*, its access networks zone). If it is not found, another DHT lookup is issued at a upper level (*e.g.*, POP zone). If it is still not found, another DHT lookup is made at a upper level (*e.g.*, domain level), *etc.* When the DHT lookup is successful at a given level, the result is returned to the client. It is to be noted, that despite the hop-by-hop routing and local resolution this provides, the top DHT level has to contain bindings for all data registered in the domain, with possible scalability and possibly performance issues. The Figure 2.1 illustrates the routing scheme of NetInf architecture.

SAIL [WWWh, SAI13] is a EU-funded research project of Network of the Future. SAIL chooses the NetInf solution as its fundamental networking architecture. The mains research directions of SAIL are: Migration, Standardisation, Business and Socio-Economics, Network of Information, Open Connectivity Services and Cloud Networking.

2.2.2 DONA

The **Data Oriented Network Architecture (DONA)** [KCC+07] project aims to define a clean-slate ICN network. The core of the routing in DONA is a hierarchical naming resolution system with a flat content naming.

The naming issue in DONA is similar as the naming of NetInf. DONA uses a flat *P:L* naming structure. The part *P* is the hash of the public key of the content owner (the Principle concept in DONA). And the *L* is the owner assigned content label. The content owners have the responsibility of ensuring the entire *P:L* name is globally unique.

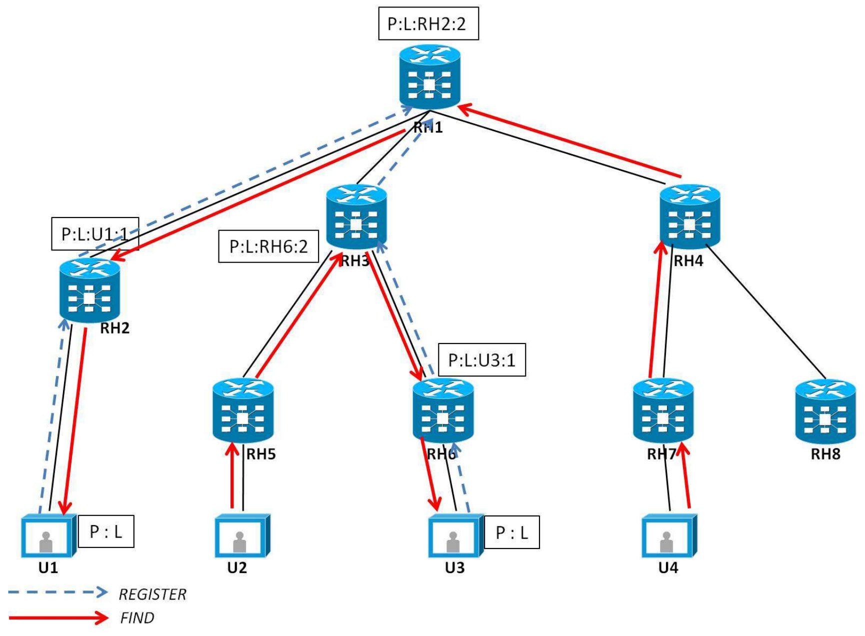


Figure 2.2: Routing scheme of DONA

However the routing in DONA is different from the NetInf routing. DONA applies a hierarchical content name resolution system (Figure 2.2), the Resolution Handles (RHs). Each RH node has a information base which contains three tuples, the content name *P:L*, the *next hop* and the *distance*. The *next hop* is from where the node receives the content name advertisement. The DONA routing contains content *FIND* and contain *REGISTER* two processes. Both of the two processes are directly based on the flat content name. When a RH receive a *REGISTER* message, it will add the <P:L, next hop, distance> into its register table for a new arrival message, or update the next and distance for an existing entry if the new arrival message has a shorter distance. Then the RH will forward this message to its parent RH(s). At last the content registration will end at the highest root RH(s). When a RH receives a *FIND* message, it will look the content name in its local register table. If it finds a matched entry, the *FIND* message will be forwarded through the *next hop* of the matched entry. Otherwise, the RH will transfer the *FIND* message to its parent RH(s). As the *FIND* message forwarding, each RH which is on the path appends the *FIND* locally, hence once the *FIND* arrives at the closest content container, the content object will be returned via the reverse path of the *FIND* forwarding.

2.2.3 PSIRP and PURSUIT

The **Publish-Subscribe Internet Routing Paradigm (PSIRP)** [ZGR+10, WWWf] is another European FP7 project. PSIRP proposed a clean slate ICN architecture which is based on a publish-subscribe solution. PSIRP applies also a *P(prefix):L(locator)* naming structure. The content name is referred as *Resource Identifiers* (*RIds*). The PSIRP network includes a concept named *Scopes*, which is identified with *Scope Identifiers* (*SIds*). The *Scopes* control the characteristics of a content, such as access right, authorization, availability, reachability, replication, persistence and the upstream resources. Both the content publication (*publish*) and the content request (*subscribe*) of a content are based on a pair <SId, RId>.

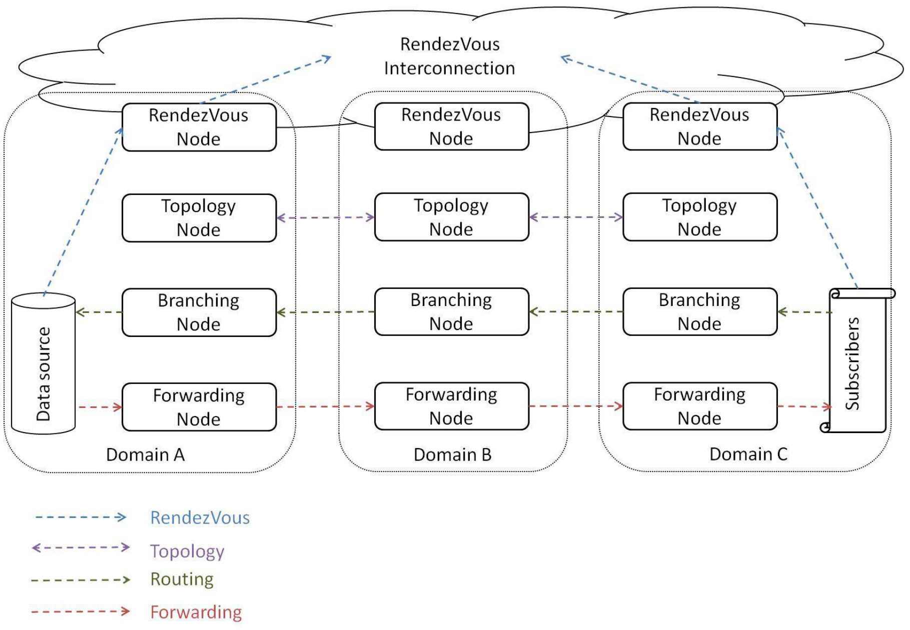


Figure 2.3: Routing scheme of PSIRP

The PSIRP routing scheme includes four important units: *RendezVous Nodes (RN)*, *Topology Nodes (TN)*, *Branching Nodes (BN)* and *Forwarding Nodes (FN)*. The entire PSIRP network is divided into *Domains*, which is similar as the Autonomous System of the current Internet. Each domain has one RN, one TN, one BN and several FN. The RN of each domain is in charge of the matching between the content publishers and subscribers, the location of the content publications and the scopes. Every individual RN can have its own name resolution system. All the RNs of every domain are interconnected with DHT into a global RendezVous Interconnection (RI) which makes the scopes of each domain globally reachable. The TN is in charge of managing the intra-domain networking topology and load balancing. It also exchanges the path vector information with the other inter-domain TNs. The BN builds up a routing map for routing the subscriber interests toward the inter-domain or intra-domain content containers by using the topology which is maintained by the TN. Finally the role of the FNs is to use a Bloom filter based forwarding implementation to realize the content forwarding from the content container to the subscribers. The Bloom filter which is named Forwarding Identifier (FId) is modified during the subscription delivery.

In PSIRP a subscriber expresses its subscription for a content to the local RN of its domain to get the content container location. The BN uses the network topology which is obtained by the TN to forward the subscription to the content container. The subscription packet injects the return path into the Bloom filter and construct the FId. At last the FNs use the FId to return back the required content to the subscriber. The Figure 2.3 illustrates the routing scheme of PSIRP architecture.

The PSIRP project has ended at 2010, but the work has been carried on in the PURSUIT project [WWWg] [PUR12], which is also a FP7 European project. This project proposed to refine the PSIRP architecture, for both wireless and wireline networks, in particular on important aspects such as caching mechanisms for better resource utilization and management, transport issues and enhancing mobility with network topologies.

2.3 Content-Centric Networking

After having analysed the ICN candidate solutions, I decided to select CCN (Content-Centric Networking) for my research target, because it is the most promising one in my opinion. Since my work relies on it, the CCN solution is much more detailed in this section than the other previously described proposals.

Content-Centric Networking [JST+09] was proposed by Palo Alto Research Center (PARC) in 2009. It is one of the most attractive ICN research candidates. From the beginning, there are many research projects on this topic in the world, such as the Named Data Networking(NDN) project [ZEB+10], the French national CONNECT research project [WWWa], European FP7 COAST project [WWWc], *etc.* The ultimate objective of CCN is to replace the IP based Internet that has met the bottlenecks as mentioned in Chapter 1, with a content based model. CCN proposes a pure content based networking architecture from content naming, content based routing, content discovery and delivery, to content based security, in-networking caching.

Everything in CCN is based on the content names. In CCN, the networking locator concept such as IP address is abandonned. The only identifier in CCN is the content name itself. In CCN the information exchanges are not based on established communication channels. The fundamental concept in CCN is that when the end-users want whichever content information, they express their requests, which contain the content names, into the network and any entity that has the matched content, based on the content name, can reply the request with the matched content object.

In the following sections I will present some technical properties of CCN.

2.3.1 CCN architecture and message exchange

In CCN, the networking elements are no longer identified by the location-related identifiers, but by the unique content names. Each networking process is also based on the content names. CCN contains two types of packet: *Interest* for sending the requests and *Data* for replying the matching content. Each *Interest* packet carries a *ContentName* (CCN name), which expresses what the client wants. Each *Data* packet carries also a *ContentName*, which is used to describe the content in this *Data* message and to match the *Interest* message. The CCN network is composed by the CCN nodes. A CCN node can be everything: a router, a switch or even an end-user terminal. CCN uses *face* concept instead of the *interface* notion which is familiar in IP. The CCN face is a relative general concept, defining not only the physical interface that connects the equipment but also the internal connection between the upper software applications and the down hardware layers. Figure 2.4 illustrates the architecture of a CCN node. Each CCN node is composed by three main elements:

1. **Pending Interest Table (PIT)** – The Pending Interest Table is introduced in CCN. It contains two information in the table: the ContentNames of the incoming *Interest* messages and the associated incoming faces. The Table 2.1 gives an example of a CCN PIT, which has two important roles. The first one is for keeping the reverse path of the propagated *Interest* packet so that the returned *Data* packet can follow these "*breadcrumbs*" downstream to the consumers. When an *Interest* packet is forwarded out, the PIT memorizes the *Interest* name and the incoming face locally (*e.g.* the third entry in Table 2.1). When a *Data* message is returned back, the PIT looks the *Data* message name up in the table and if there exists a matched *Interest* name, the CCN node will send the *Data* packet out through the associated face(s) of the matched entry. The second role of PIT is for avoiding multiple request forwarding. When the *Interest* messages for the same content but from different incoming faces are received, only the first one is forwarded, the others will be only appended in PIT and waiting for the *Data* message back (*e.g.* the first entry in Table 2.1). Once the CCN node receives a replied *Data* message from the first sent out the *Interest* packet, all the other appended *Interest* messages can get a copy of the *Data* message so that each client can receive the required *Data* object.

CCN Pending Interest Table	
Content names	**Faces**
ccnx:/youtube.com/news/baby_born/	face308, face321
ccn:/google.fr/	face103
ccn:/orange.fr/news/meteo/	face201
. . .	. . .

Table 2.1: CCN Pending Interest Table

2. **Forwarding Information Base (FIB)** – The CCN FIB is similar as the FIB in IP routers. It contains the ContentNames or the aggregated prefixes of the names, and the outgoing face. Table 2.2 shows an example of a CCN FIB.

FIB is used for forwarding the *Interest* messages to the sources that are known to potentially hold the matching *Data* message. For example if a CCN node receives an *Interest* message on a name as *ccnx:/youtube.com/news /baby_born/*, and if the Content Store does not have a matched content and the PIT does not give a matching result, the *Interest* packet will be forwarded through the *face101* according to the FIB. The IP FIB is filled by the routing messages. The CCN FIB is filled by the content advertisements. If a content provider wants to publish some available contents, it should send out the content advertisements into the network. Every CCN node that receives such a content advertisement will add the content name or prefix together with the incoming face identifier in the FIB. An entry of CCN FIB can have multiple outgoing face, that means the multicast is supported by default in CCN.

CCN Forwarding Information Base	
Content names/prefixes	**Faces**
ccn:/youtube.com/	face101, face102
ccn:/google.fr	face103
ccn:/orange.fr/news/meteo/	face201
. . .	. . .

Table 2.2: CCN Forwarding Information Base

3. **Content Store (CS)** – CS is a cache or a buffer set in every CCN nodes. It can cache the content objects that pass by and provide the in-network caching feature for the goal of minimizing network bandwidth and latency demands, as well as the server load. Content Stores of different CCN nodes can have different caching strategies and content replacement policies. The Content Store is an important element in the CCN network because it provides the on-path caching benefits. However in the CCN research world there currently have two different understanding of the Content Store. Some people consider the Content Store as a CDN-cache like data base, that means the Content Store can have a large storage volume and provide the CDN-like service. The others think that the Content Store is only a short on-chip buffer. The main functionality is to provide a fast retransmission reaction in case of a packet loss. In my thesis I am aligned with the second consideration. Because first of all, the on-path storage requires fast performance, however the current fast memory technology does not support large volume. Secondly whether deploying the large CDN-like caches everywhere on-path of the CCN network can really offer a good performance than the conventional off-path CDN service is not clear.

The packet process in a CCN node is performed as following. When an *Interest* packet arrives at a CCN face, the ContentName carried by this *Interest* packet is firstly checked in the Content Store. If there is a matching *Data* message, it will be directly returned through the same face where the *Interest* message arrived at. Otherwise the ContentName is further checked in PIT. If there is already a matching entry, the arrival face information is updated in the matching entry. If not, the FIB

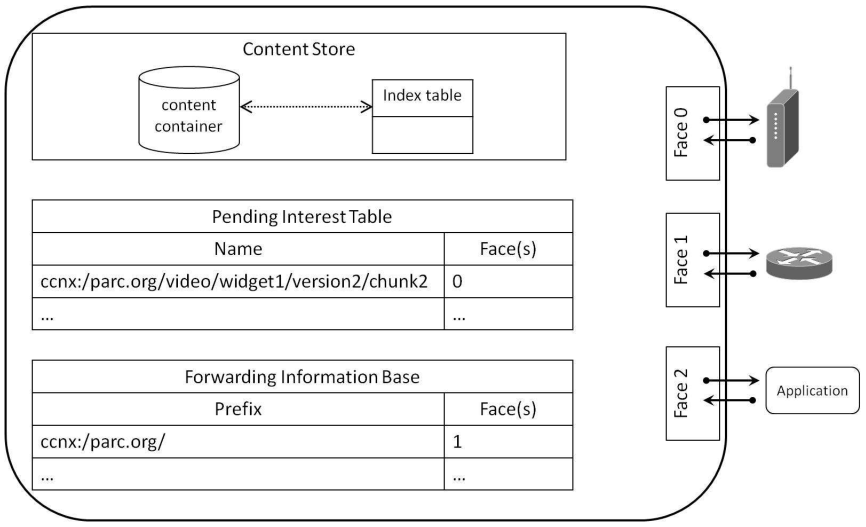


Figure 2.4: The node structure of Content-Centric Network

table is consulted for the forwarding information, and a new entry associated with the new arrival *Interest* message and its incoming face is created in the PIT.

The *Data* packet follows the tracks left by the *Interest* packets in the CCN nodes. When a *Data* packet arrives at a CCN node, the ContentName of the *Data* packet is checked in the Content Store. If there is already a matching content cached in the CS, this *Data* packet should be discarded. Otherwise, the *Data* message is checked in PIT. A PIT entry match means this *Data* message has been required, it is then sent out through all the faces that are listed at the matching PIT entry, and the *Data* message can be (optionally) stored in the Content Store based on the caching strategy.

2.4 CCN Naming and Routing

In CCN everything is based on content names. The naming and the named-based routing scheme are the core of all the activities in CCN networking. The content retrieval in CCN can be mainly divided into two parts: the content discovery and the content delivery. The content discovery is related to how a content is named, how it is published and how a CCN node addresses it. The content delivery defines the CCN routing protocol which is about how a content provider propagates its contents into the network, how a CCN node routes the end-users' *Interest* packets to the best content sources and how a CCN node delivers the contents to the end-users.

The content name is the only identifier of each content object, which permits either the end-user or the intermediate networking unit to locate the best content holder. The content name is usually a globally unique identifier, but the unique named content can be in different containers, for example the original content servers, the CDN repositories or the on-path caches.

The content name in CCN is designed as a hierarchical structure (Figure 2.5). The hierarchical name is organized as a *prefix-suffix* order. *ccnx:/parc*

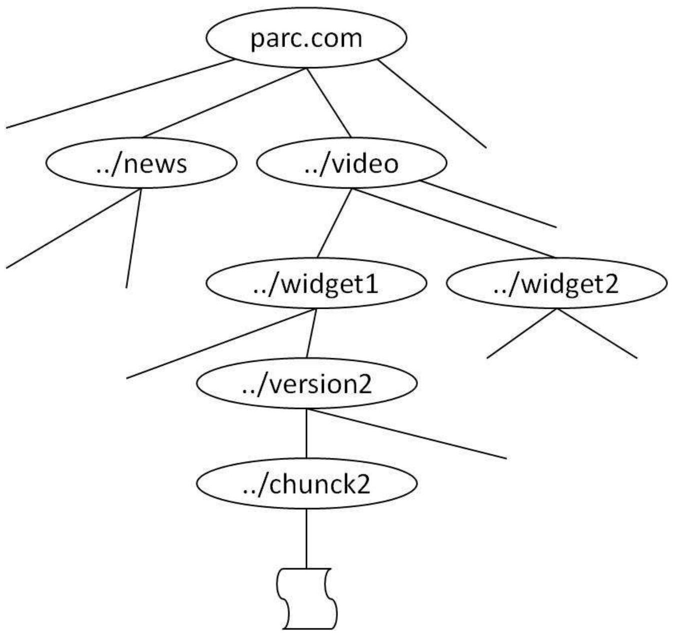


Figure 2.5: The naming of Content-Centric Network

.org/video/widget1/version2/chunk2 is an example of a CCN content name. All the content provided by *parc* can share the same *ccnx:/parc.org/* prefix. The tree-like structure can make the CCN name support the aggregations in CCN FIB as the IP address aggregation. A same information object may have several different versions and a single content can be divided into multiple small segments (*chunks* concept in CCN) in order to adapt the transport layer. Thus each CCN name is ended by the version and chunk information which can simplify the content discovery. The entire name is signed with a SHA256 digital signature of the content provider. Thus only the authorized receivers can decrypt the digital data.

The CCN routing is realized via direct name based routing. However, as I presented in 2.3.1, in CCN only the *Interest* messages are routed, but not *Data* packets. The CCN FIB is filled with the content advertisements that are flooded by the broadcasting protocol, for example the OSPFN protocol [WHY+12] as proposed by the NDN project. OSPFN is the first relatively mature proposal related to content advertisement in CCN. I present some details of this protocol in the next subsection.

2.4.1 OSPFN protocol

The NDN project [ZEB+10] proposes OSPFN [WHY+12], which is an extended version of the IP OSPF [Moy98] protocol for CCN, to broadcast the content advertisements. OSPFN uses the Opaque Link State Advertisement (OLSA) to announce content names or prefixes and router IDs. When a CCN node wants to publish an advertisement, it creates an OLSA message which contains the name of the content that it wants to publish and its router-ID. Its OSPFN engine is in charge of flooding the OLSA message over the network in the same way as the IP OSPF protocol. Another CCN node who receives this OLSA message will retrieve the content name and the router-ID information that are carried in the message and query its local OSPF engine with the router-ID to calculate the shortest path to the publisher and find out the right next hop. Finally, the CCN engine injects the content name and the next hop in its local FIB.

2.5 CCN on-path caching

One of the most remarkable characteristics that differs CCN from the current IP Internet is the on-path caching system, which is offered by the Content Store design. As I described in Section 2.3.1, the Content Store can be implemented in various types of CCN nodes (*e.g.*, routers, end-user terminals, border gateways, *etc.*). Each Content Store can have the different caching and replacement strategies according to the ISP needs. The Content Stores at different nodes works individually in the original design. In order to get a better performance for different services, others caching strategies can also be applied (for example a cooperative strategy [LS11]).

CCN protocol takes advantages of its URL alike hierarchical content name and a multicast routing mechanism to forward user's request to multiple sources of the content. The source could be the original provider of the content, or the users who are willing to share their local copies of it. Any network nodes along the path from the requester to the source holding the corresponding content can directly satisfy the end-user's request. At the meantime, the node that do not cache can decide to store the content passing through it according to a certain policy. Since the copyright check and the authentication of the content is accomplished at the application level, network operator does not need to implement specific functions to manage the storage space at user side and authorize the publication of the content from them. Because of the same reason, the CCN protocol deals with the cache storage offered by othe network operator and contributed by the users in the same manner. The CCN name-based routing is simple and efficient enough to retrieve the cached content.

2.6 Summary of ICN state-of-the-art study

The idea of Information-Centric networking is widely discussed in the research world but it is still at a earlier stage and many features are still open for discussion. The just presented ICN solutions have their own advantages and drawbacks. They can be seen competitors or complementaries, depending on the objectives. I made a brief summary of the key features of these ICN solutions in Table 2.3.

In the beginning of my thesis, I made an analysis of those solutions to decide which solution to take into consideration for my research work. I analysed them based on several features, such as scalability, naming, routing, forwarding, mobility support, the potential to be deployed by network operators and the possible networking usages. The following text detail the main reasons why I finally concluded that the CCN solution is the most suitable.

The CCN URL-like hierarchical naming is flexible for the content names management. The prefix parts of a name can used for distinguishing different content sources,

Solutions	**Naming**	**Forwarding**	**Caching**	**Mobility**
CCN	Hierarchical naming	Name based routing; Longest prefix matching	In each CCN node	Transport-based native support
DONA	Flat naming	Pub/Sub system; Label switching	Only in Resolution Handler node	Not mentioned
PSIRP	Flat naming	Pub/Sub system; Label switching	Only in RendezVous points	Not natively support
NetInf	half-flat and half-hierarchical	MDHT overlay	In network nodes	Overlay-based support via NRS

Table 2.3: Comparison of existing ICN solutions

content or service types or even including the administrative information. The unbounded feature can support the self and dynamic content name generating. The longest prefix matching routing allows the users to query the contents without knowing the full names, or even the content which are not yet published. The URL-like naming and the aggregatable prefix are very similar as in IP, so that some mature IP technologies can be reused for CCN, such as the BGP or OSPF protocols. Thus, if in the future the IP based network and the ICN will temporally co-exist and cooperate with each other, CCN is better for the migration.

CCN forwarding performs a direct name based routing, without adding any third party element (contrarily to the naming resolution system in NetInf or the centralized RendezVous points in DONA). The DNS service for IP is now too heavy to manage and is already vulnerable, for the security issue. The direct name based routing from CCN is lighter for the realistic implementation and it is less complex to adapt CCN into various networking environments, such as the Wireless Sensor Network, the vehicular networks or domestic networks.

CCN does not differentiate a lot between the networking level router and the end-user level terminals. Each equipment in the network is a CCN node. So that the CCN in-network caching features can be carried out in every CCN unit. Since the design of the centralized caching node will suffer the capacity or the management issue, the caching structure from CCN can extend the adaptability of the caching feature for various networking use cases.

The mobility becomes an important issue for current networking performance. The predominant current Internet applications, for example streaming delivery, security, VoIP, cannot still deal with the mobility requirement. Even if the mobility in CCN is not yet mature, the location-independent networking design and the no-connection based datagram transmission mode will have a better opportunity to deal with the mobility issue.

Based on the aforementioned points, I chose CCN as the ICN solution in my thesis research work. A scalable issue of how to manage billions content names is unavoidable for the content based CCN and, for example, leads to memory issues in the local nodes, because the memory requirement of the naming tables will be very huge. This is the starting point of my research work and the related challenges I decided to address.

2.7 CCN optimization and Memory Technology

As I mentioned in the previous subsections, the CCN paradigm proposal can bring many benefits but it is also a big challenge for current hardware technology, especially the memory requirement issues. The in-network caching design requires a large memory space for content storage (Content Store). The switch from IP addresses to the content names will also introduce a huge number of content identifiers, that requires a large memory space to memorize the content names locally in each CCN node for the information forwarding (PIT and FIB). And according to the performance requirement of content caching or packet routing, the implemented memory chips should perform fast.

Technology	**Access time(ns)**	**Cost($/MB)**	**Max. size**
TCAM	4	200	$\approx 200Mbits$
SRAM(on-chip)	1	27	$\approx 50Mbits$
SRAM(off-chip)	4	27	$\approx 250Mbits$
RLDRAM	15	0.27	$\approx 2Gb$
DRAM	55	0.016	$\approx 10GB$
High-speed SSD	1,000	0.03	$\approx 1TB$

Table 2.4: Current memory technology [PV11]

Some related papers have already expressed some doubts about whether the current hardware can support the novel content centric paradigm [PV11] and the realistic implementation issues of CCN. Lee *et al.* from [LRH10] analyzed the energy efficiency of CCN content storage. Rossi and Rossini from [RR12a] studied the caching performance of CCN on the size of individual router caches. Somaya *et al.* from [ANO10] investigated a CCN router design and implication of the CCN content store. The above analysis research works are all based on the current CCN structure without proposing an optimizing design which can improve the performance. And furthermore, most of the works are all about the CCN caching element (Content Store), but less on the forwarding units (PIT and FIB). I tried to fill this gap and evaluated this memory issue for the FIB and PIT CCN components. Looking at Table 2.4, which summarizes today's memory chip technologies, we can make a quick computation to highlight the limits of current memory technologies for CCN. Let us suppose that the average length of ContentNames in the *Interest* packets has 40-Bytes, that a CCN node receives an average *Interest* packet arrival rate as 100Mpck/s, that the average *Data* packets response time is 80ms. According to the *Little's Law*, we need more than 2Gbits memory space for the PIT and FIB. Even if, in the FIB, the content names can be aggregated into shorter prefixes, the required memory space is still in the order of Gigabit. The 100Mpck/s rate needs at least the SRAM to meet the access time speed, but the SRAM can not support such a big volume requirement.

This example, showing the limitation of memories regarding the storage size versus the access time, has driven my contributions in this thesis, which is how to optimize the CCN node design in order make it meet the current hardware conditions.

Chapter 3

DiPIT: A Bloom Filter based CCN PIT redesigning

3.1 Introduction

The Content-Centric Networking proposal implies a major shift in the design and the implementation of a new information-oriented networking. The name based paradigm can bring many benefits, as mentioned in Chapter 2, but it is also a big challenge for today's hardware technologies. For example, the name based design will create plenty of different content names in the network, and lead to one challenge related to the memory requirement for a realistic implementation of the CCN architecture. Table 2.4 shows a summary of today's fast memories. We can see that the fast chips (*e.g.* TCAM or SRAM), which have a shorter access time, does not support a large volume, and a large chip (*e.g.* RLDRAM or DRAM), which can support larger space requirement, does not provide a fast access. Some researchers have argued that today's router technologies do not meet the requirement of CCN nodes [GKR+11a, PV11]. These analysis however assumed a same node architecture than the classic one described in the first CCN paper [JST+09]. My work in this chapter aims at developing a new CCN node implementation, which not only preserves the original features of CCN, but can also realistically be deployed with current hardware and software technologies. As such, I propose a new implementation for one of the main components of the CCN node, the *Pending Interest Table* (PIT).

The PIT is a central component in a CCN node because it is involved in every message processing. However it has not get too much research attention up to now. Indeed, previous works related to CCN have more focused on the Content Store [ANO10, MCG11] or on memory architecture for efficient FIB forwarding processes [HAM11]. The role of a PIT is to store *Interest* packets that passed through a CCN node until these *Interest* messages are "fulfilled" by the reception of matching *Data* packets. For every reception of *Interest* packet, the PIT should create or update an entry, which contains the *Interest* name and the incoming face information. For every *Data* packet, the PIT should find out the matching entry and retrieve the outgoing faces for *Data* packets delivery, and then delete the matching entry.

In an IP network, the size of the FIB table is relative stable and independent from the packet arrival rate. But the CCN PIT is different (see Chapter 2 for details). According to the highly dynamic behaviour, the implementation of the current PIT design cannot accommodate to current memory technologies. A PIT needs a large memory space to store the pending *Interest* packets (PIT cannot aggregate on content name prefix because *Data* packet and *Interest* messages should exactly match between each other). Considering an average *Interest* packet arrival rate of 125 millions packets per second (hereafter noted *Mpcks*) and a packet round-trip time of 80 ms [PV11], the number of the *Interest* messages which should be stored in PIT can easily reach the order of 10^7. In addition, the content name structure is longer and more complex than the IP address, even IPv6 [GKR+11b]. Therefore the memories with large volume should be used, with the known limitations, especially long access time. Contrarily to traditional IP FIB tables, a PIT table in a CCN node is highly dynamic. For every incoming *Interest* packet (unless the requested content has been already cached locally) and every matching *Data* packet, an operation should be carried out in the PIT table. Consequently operations should be performed fast, which requires fast memories that are unfortunately only available for small storage size.

To tackle this inextricable problem, I propose to distribute the single centralized PIT of a CCN node to several sub-tables associated with each face, which I call *PITi*. Each PITi has relatively smaller size, and it does not need to remember the incoming interface information. Thus it is possible for the fast memory to support one PITi table. I also implement each PITi with a counting Bloom filter, that can largely reduce the total memory space. Benefits of this system include fast lookups and small storage requirements. My distributed PIT proposal (namely *DiPIT* for Distributed PIT) is independent from the naming structure, whatever the naming is hierarchical or flat, long or short. In this Chapter I first analyse the limitations of the centralized hash table in condition of today's memory technologies. After that I give a general description of the distributed PITi architecture. Then I give some evaluations to analyse the distributed PIT solution performance in the terms of memory space consumption and the networking load aspect. Finally I propose an example of a hierarchical network with such a distributed PIT system.

3.2 Background and Related Works

Up to now, previous work related to Content-Centric nodes have mainly focused on the performances of Content Store [ANO10, CGP11, MCG11] and on memory architecture for efficient FIB multicast forwarding processes [HAM11]. Indeed, researchers are familiar with those domains of activities, since the Content Store component is functionally close to the existing caches or the CDN nodes, and the CCN FIB is similar to the IP FIB table. But the design and implementation of the PIT table have not get much attention so far since the PIT is a completely new component, not present in IP or in others information-centric networking proposals [ZGR+10, KCC+07]. However, this component is at least as important as the Content Store and more demanding than the FIB table because of the frequent updates in the PIT, contrarily to the FIB. This is the reason why I precisely focus on this neglected component that plays a crucial role in the performance of the CCN node.

In order to improve the CCN PIT lookup performance, I started my research by studying the conventional lookup techniques in IP networking.

3.2.1 IP lookup

There are mainly four lookup techniques in IP network: TCAM [RM04, LS10], radix tree [Skl91, LKN13], hash table based [WVTP97, TL11] and Bloom-filter+hash-table solution [DKT03, YMB09]. As discussed in [PV11], none of them is acceptable for a CCN node. TCAM is a hardware solution with bounded lookup performance at one single memory access. It is perfect for a *one-in-multiple-out* lookup. But TCAM is expensive and the memory space capacity is limited. Radix tree is not suitable for the CCN naming structure because a naming tree willbe very wide and deep. Hash table can perform fast lookup operations but, since CCN PIT is based on exact matching, every entry should be memorized. If we do not want to have a big collision rate, large memory is required. Finally, Bloom-filter/hash-table implements the Bloom filters on fast memory chips as a pre-verification step for the purpose of reducing the number of accesses on the slow hash table. But since the injecting step of the Bloom filter breaks the original data structure, it can not directly be employed in IP FIB. So the hash tables are still required for prefix. information retrieval. Then this approach suffers from the same drawback as the previous approach: the aggregation is still needed and the memory requirement for the hash table is too big for fast memory chip.

Even the Bloom-filter/hash-table method is not suitable for the CCN lookup, but the fast verification feature of Bloom filter solution still got my attention. There are also several studies on the implementation of pure Bloom filter structure for the IP lookup. The BUFFALO system [YFR09] distributes IP forwarding tables on each router interface. Since this system does not tolerate too many entries, it targets especially small-scale networks (typically enterprises) and data-center networks. More recent papers dealing with Bloom filters for IP lookup or routing (for example [SHKL09, JZER$^+$09, JF08]) assume a relatively stable set of elements, which is different form the dynamic behaviours of the PIT.

3.2.2 Bloom filter

Th Bloom filter was first introduced by Burton Howard Bloom in 1970. It is a space efficient probabilistic data structure which can be used to test the existence of an element in a set. The idea of a basic Bloom filter is to store the footprint of every element in a vector by using just a few bits positions.

A Bloom filter includes a long binary vector and several independent hash functions. The Figure 3.1 illustrates an example of a binary Bloom filter. To create a Bloom filter, every member of the set (the *ccn:/youtube.com/video1*; *ccn:/lemonde.fr/text1* and *ccn:/orange.com/live/stream1* in Figure 3.1) should be passed one by one through k (k = 3 in this example) different hash functions. Each hash function will code this element in a position of the vector whose length is m. Despite of the collision, every element has k position bits in the vector. When we check if an entry in the set (for example the *ccn:/sina.com/*), we also pass this element through the k

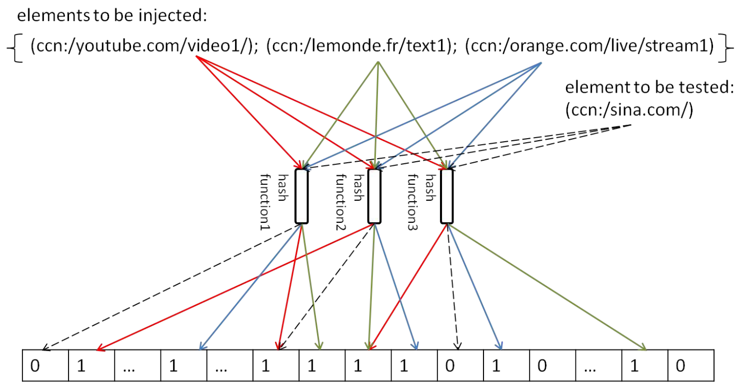


Figure 3.1: A binary Bloom filter structure

hash functions, if all the positions give positive answer, this entry can be considered as exist in the set, otherwise, this entry definitely does not in the set.

Indeed, the Bloom filter can introduce the false positive because each position bit can be set to 1 when adding other elements. If we note the number of members in the set as n, the vector ability is m bits and we use k hash functions, the possibility of the false positive can be calculated as :

$$fp = (1 - e^{\frac{-n \cdot k}{m}})^k \tag{3.1}$$

With given m and n The optimal value of k is:

$$k_{opt} = \ln 2 \cdot \frac{m}{n} \tag{3.2}$$

In this condition, the possibility of the false positive is in the minimal value. The following figures show the evaluations of the false positive possibility in function of the ratio of m to n from 1 to 50. The first four figures give the evaluation with $k =$ 3, 4 and 5, and the following one is with the k_{opt}.

The figures in Figure 3.2.2 clearly show that when m/n is big, for example bigger than 6, the fp(k=5) < fp(k=4) < fp(k=3). But when m/n stays in the area (1:5), fp(k=3) < fp(k=4) < fp(k=5). If we take a look at the difference of false positive at each m/n between k = 3,4,5, we can see that when m/n is bigger than 5, the difference between the maximum and the minimum is always less than 2%. To conclude of this section, to make the false positive less than 10%, the ratio of m to n should bigger than 5, and when m/n is bigger than 5, the difference between k = 3 or 4 or 5 is not very important, even compared with the optimal k, we can still choose k = 3 for saving the memory access time.

3.3 Preamble: Modeling PIT table

I provide in the following a model analysis of the CCN PIT table. I also introduce the main notations that I used throughout this subsection in Table 3.1.

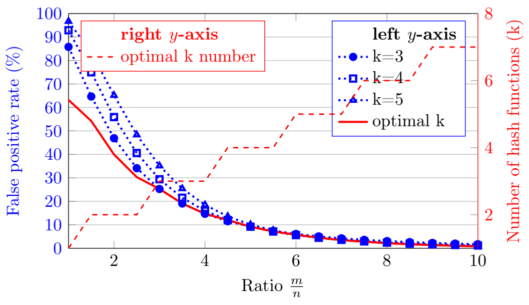


Figure 3.2: k=(3;4;5), m/n=(1:10)

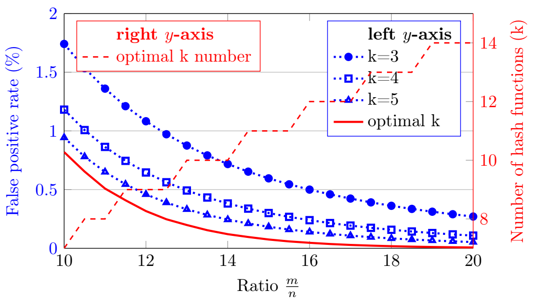


Figure 3.3: k=(3;4;5), m/n=(10:20)

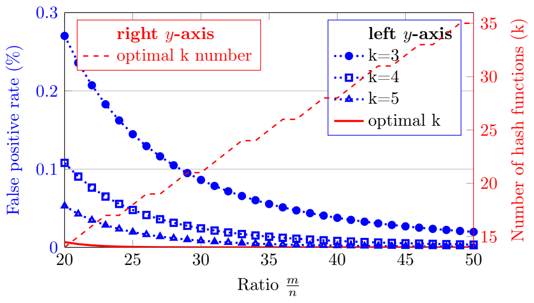


Figure 3.4: k=(3;4;5), m/n=(20:50)

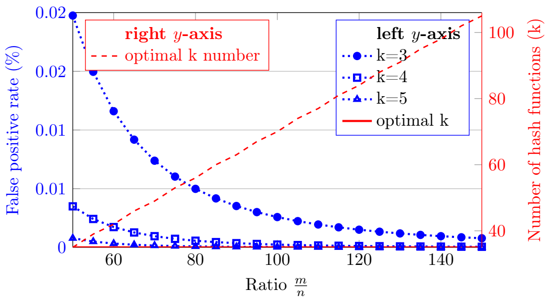


Figure 3.5: k=(3;4;5), m/n=(50:150)

When an *Interest* packet arrives a CCN node, it should first be checked in Content Store. This *Interest* packet is then considered by the PIT table if no *Data* object cached in the Content Store can fulfill this request. Thus the *Interest* packet entering rate for the PIT is:

$$\lambda_{pit} = \sum_{i=1}^{nbr_{face}} \lambda_i \cdot (1 - CS_{hit})$$

The entries in PIT table are then "consumed" by the *Data* packets, so the number of entries is in average $\lambda_{pit} \cdot RTT$. We also consider redundant *Interest* packet, thus we obtain that the actual number of stored PIT entries is:

$$Nbr_{pitEntry} = \sum_{i=1}^{n_{face}} \lambda_i \cdot (1 - CS_{hit}) \cdot RTT \cdot \tau_{interest} \tag{3.3}$$

3.4 Analysis on the centralized hash table

The PIT table described in the seminal CCN paper [JST+09] is a centralized hash table. This implementation is also chosen by default in the open-source release CCNx [WWWe]. I analyze in this Section the scalability of the hash-based CCN PIT table. In the evaluations I consider only the two fastest memory technologies, the SRAM and the RLDRAM. The SRAM could be built either on-chip or off-chip

Parameters	**Meaning**
n_{face}	number of interfaces in the CCN node
λ_i	*Interest* packet arrival rate at the interface i
λ_{pit}	*Interest* packet entering rate to the PIT table
RTT	average *Data* packet round-trip time
CS_{hit}	hit ratio of the Content Store
$\tau_{interest}$	ratio of *Interest* packets for the same ContentName. It depends on the traffic distribution
$Nbr_{pitEntry}$	number of stored entries in the PIT table

Table 3.1: CCN node modeling parameters

(Table 2.4). The on-chip SRAM memory offers a faster access time however it is limited at 50 Mbits size. This size is too small for the PIT table, so SRAM refers to the off-chip SRAM in the rest of the paper.

I highlight in Equation (3.3) that the number of PIT entries depends on both CS hit ratio (CS_{hit}) and traffic popularity distribution ($\tau_{interest}$). Both parameters do not vary much for a given CCN node, and independent of the implementation design of the PIT. To simplify the analysis, I set $CS_{hit} = 0$ and $\tau_{interest} = 1$.

3.4.1 Table size and Cost

Each entry in a hash table PIT contains the key value of ContentName (of length $size_{key}$) and the incoming interface identifier information (of length $size_{face\_id}$). The estimated hash table size is then:

$$Size_{hash} = (size_{key} + size_{face\_id}) \cdot \sum_{i=1}^{n_{face}} \lambda_i \cdot RTT \tag{3.4}$$

In the evaluation, I suppose a router with $n_{face} = 4$ interfaces. The parameter λ_{in} ranges from 0 to 200 Millions of packets per second (Mpcks). The average packet RTT is 80 ms as defined in [PV11]. In order to hash the content names, we use three different hash values: 24 bits, 32 bits and 64 bits.

In Figure 3.6 and Figure 3.7 we present the cost of implementing a hash-based centralized PIT table on a SRAM memory or on a RLDRAM memory, respectively. We can see that in Figure 3.6, all the three curves stop at *cost* = \$6,750, which refers to the maximum size of SRAM (250 MBytes). I highlight here that a SRAM is fast enough for the processing of packets, but it cannot store the entries for high *Interest* packet arrival rate. Even for short hash value like 24 bits, the SRAM can only support up to 150 Mpcks packet arrival rate. The three curves in Figure 3.7 stop at $\lambda = 66$ Mpcks since the RLDRAM cannot process more packets per second. If the size is no more a problem, RLDRAM is too slow for most CCN nodes.

To summarize this first evaluation, only SRAM with hash values lower to 32 bits can meet both packet arrival rate and memory size requirements of a high-level CCN node.

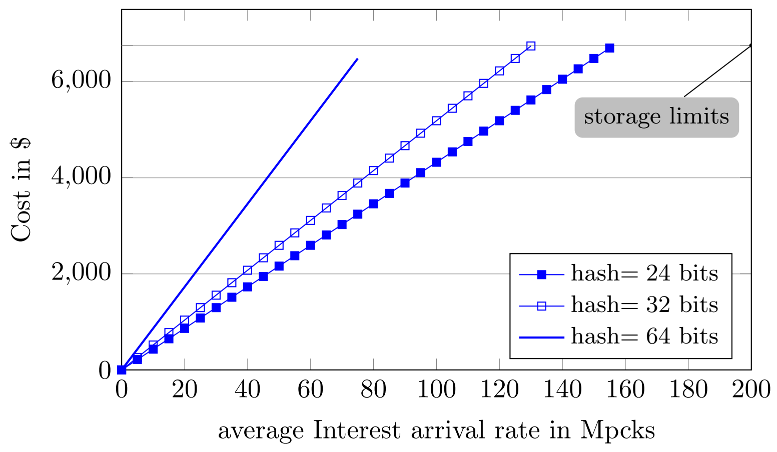


Figure 3.6: Required SRAM memory cost vs. λ_{in}

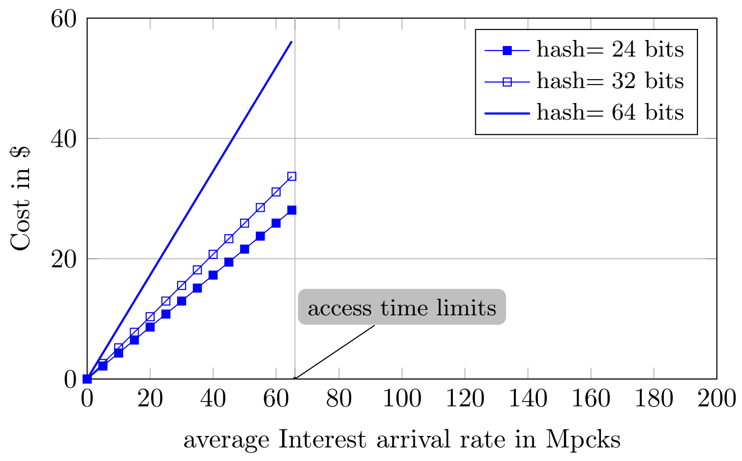


Figure 3.7: Required RLDRAM memory cost vs. λ_{in}

3.4.2 Collision ratio

Short hash value length enables the implementation of PIT with large storage of *Interest* messages. But the shorter is the hash value length, the more probable are collisions. A collision in hash table occurs when distinct elements are coded at the same entry of a hash table. The elements that suffer from a collision are stored in a linked list at the same hash entry, therefore a collision does not cause packet loss (except if the memory is full), but information retrieval is much longer. Here I evaluate the collision rate for different hash value lengths.

$$Collision = \begin{cases} \dfrac{nbr_{in} - 2^{size_{key}}}{nbr_{in}}, & nbr_{in} > 2^{size_{key}} \\ 0 & nbr_{in} \leq 2^{size_{key}} \end{cases}$$

$$\text{where } nbr_{in} = \sum_{i=1}^{n_{face}} \lambda_i \cdot RTT \tag{3.5}$$

Then I evaluate the collision ratio for hash value lengths 16 bits, 24 bits and 32 bits (since we observed that higher values are not possible). In Figure 3.8, both 16 bits and 24 bits experience collisions for small *Interest* packet arrival rate. Moreover collision ratios quickly reach their maximum level. Only the 32 bits hash values present low collision rate until 200 Mpcks.

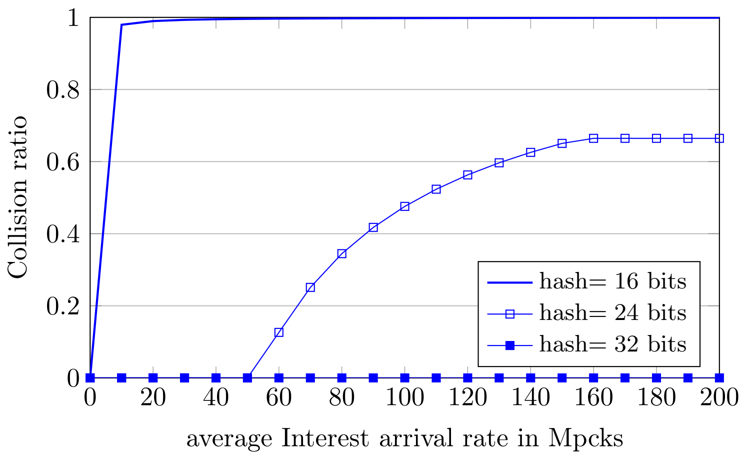


Figure 3.8: Predicted collision ratio of SRAM in function of λ_{in}

3.4.3 Discussion

Through both evaluations in §3.4.1 and §3.4.2 I observe that the selection of the memory technology depends on the packet arrival rate. For example, a RLDRAM with a 32 bits (or more) hash value can support an edge router with a low packet arrival rate (e.g. less than 66 Mpcks). However, for a bigger router (e.g., a core network router), the packet rate is generally higher than 66 Mpcks and a SRAM with a 32 bits hash value is the only option. If the rate is higher than 130 Mpcks, the hash table implementation has some critical limitations because neither SRAM nor RLDRAM can support both memory and access time requirements. Alternative solutions should be designed.

3.5 DiPIT – Distributed Bloom filter based PIT architecture

I aim to implement a fast, space-efficient and cost-friendly PIT tables, with regard to the role of PIT defined in the CCN proposal. The two most popular table implementations that enable lookup in a time that does not depend on the size of the table (in other words, $\mathcal{O}(1)$-time lookup) are hash tables and Bloom filters. For DiPIT, I chose Bloom filter solution because Bloom filters are more efficient in memory space than hash tables, that is the main difficulty. Bloom filters can also be faster than hash tables for lookup and update operations. In this chapter, I explore the opportunity to leverage the appealing characteristics of Bloom filters for the implementation of PIT table.

The traditional design and usage of Bloom filters can not be directly implemented into PIT. Because a Bloom filter only remembers the *footprint* of each element. Once an element has been injected into the filter, it is impossible to retrieve any other information again. On the contrary, a PIT, as it has been conceived in CCN, the role requires the PIT to not only check if an *Interest* name is present in the table, but also retrieve the matching facing identifiers for sending *Data* packet. This mismatch is tackled by implementing a distributed PIT table, which is named DiPIT.

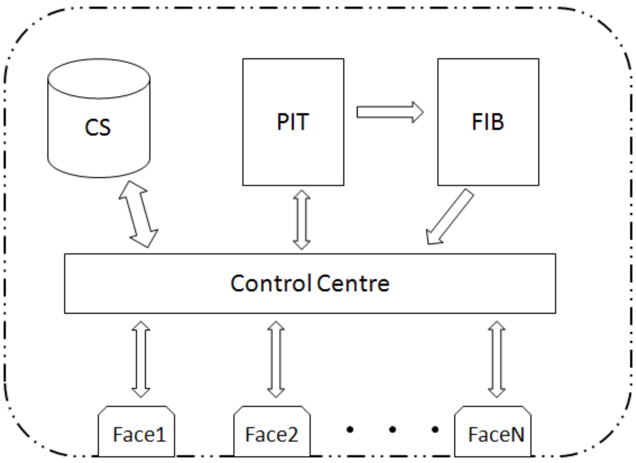


Figure 3.9: The conventional CCN node architecture

The original CCN node architecture (Figure. 3.9) implements a centralized PIT table. The DiPIT proposal constructs one PITi (a small PIT table) on each CCN face (Figure. 3.10). Each PITi is constructed by a Bloom filter. In order to further reduce the false positive rate, the system also includes one additional Bloom filter which is shared by all faces. The principles of this proposal are presented in the following.

3.5.1 PITi: One Bloom filter per CCN face

Each PITi works independently and records in a *counting* Bloom filter the footprints of the incoming *Interest* packets that come from the associated face. Incoming *Data* packets are checked in parallel on all PITis and are forwarded on the faces when

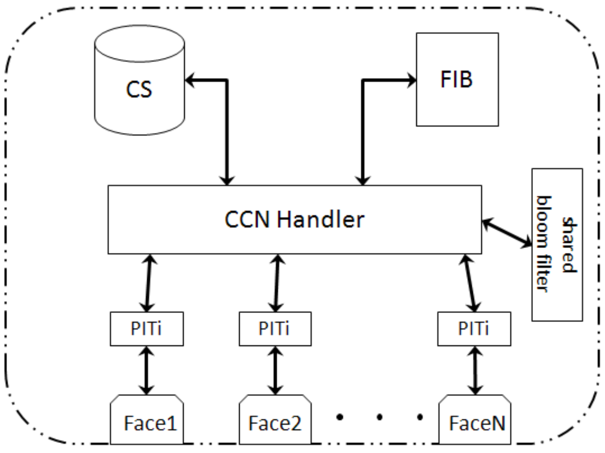


Figure 3.10: The CCN node architecture with DiPIT system

the associated PITi has a matching footprint. Thus the DiPIT system keeps the advantages of Bloom filters in terms of memory space and process time, and the design of one PITi per face tackles the aforementioned face information issue.

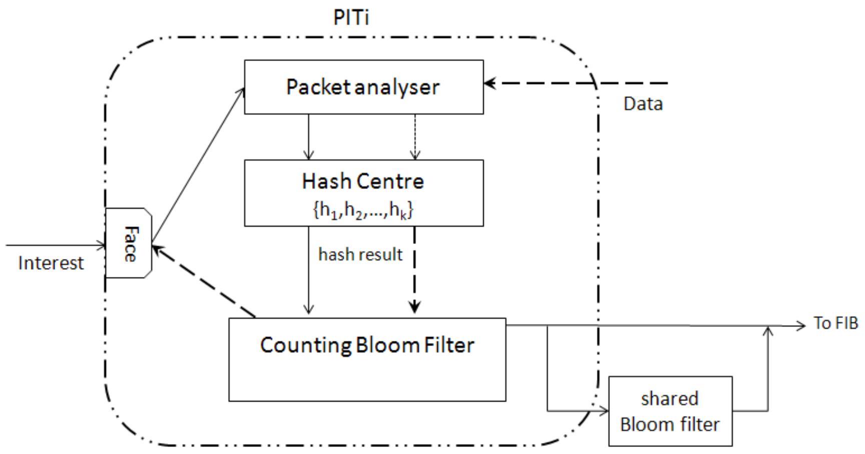


Figure 3.11: Internal PITi architecture

I use a counting Bloom filter to deal with information removal. Indeed, a binary Bloom filter does not support the removal of an element because a bit position in the vector can be set to one by more than one element. In CCN, when a *Data* packet is forwarded, the related *Interest* message entry should be deleted from the PIT table. Therefore, entries are frequently added *and removed* in a PIT. The counting Bloom filter addresses information removal by replacing the binary filter of Bloom Filters by a set of counters. Each position of the filter (each hashed position) is an integer that should be increased by one when another *Interest* packet comes. When a *Data* packet is forwarded through a face, the counters associated with the according *Interest* message should be decreased by one. The implementation of this counting

Bloom filter requires some extra memory, however it is still less than the memory space required by a hash table. I prove this by a series of evaluations in Section 3.6.

The expired *Interest* message should be removed from the PIT table. The hash-based PIT applies one timer for each PIT entry to delete the timed-out *Interest* messages. Similarly, in DiPIT I introduce one timer that manages the removal of expired *Interest* messages. Every PITi counters will be decreased by one (if not already at 0) at periodic interval. This interval is configurable and named DiPIT_TTL. This value should be large enough, *i.e.* at least twice the average response time, representing the time to wait for the reception of the matched *Data* object. If the *Data* packet is received before DiPIT_TTL, the counter will be "normally" decreased because the *Interest* message has received the corresponding *Data* message. If no *Data* message is received after the DiPIT_TTL period, the counter will be decreased meaning the removal of the expired *Interest* message.

3.5.2 A shared Bloom filter to deal with false positive

A Bloom filter can introduce false positive since each filter position can be incremented by different element insertion. The false positive rate can be estimated according to Equation 3.1, and in our case the inserted elements number n the product of the packet rate λ and the RTT.

The event of the false positive in this study is that the ContentName verification gives a matching result but in reality no *Interest* packet for this ContentName has been received on this face. We identify two side effects here.

- If an incoming *Data* packet mistakenly generates a match on the PITi of one face, this *Data* packet is forwarded through this face, although no *Interest* message matches this *Data* packet. This false positive produce some useless networking bandwidth waste. It will also delete the "fake" matching entry when the *Data* packet goes out. Thus when the actual *Data* packet comes, it might be dropped due to the missing *Interest* information. However these two problems are not critical. First of all, the extra emission will affect nothing but some networking resource consumptions. As the actual false positive rate is small, the extra emission is also limited. Secondly the probability that a series of consecutive linked routers generate a false positive on the same *Data* packet is low, so the extra networking load can be further limited on a few hops. For the mistaken deletion of a matching *Interest* message, the CCN designs incorporates regular *Interest* packet re-emission from the client side, so this event can only add some extra-latencies to the request.

- If an incoming *Interest* packet mistakenly generates a match on the PITi of one face, this *Interest* packet is considered as a duplicate thus it is not forwarded. The impact of false positives is critical because this *Interest* packet is lost.

I address the issue related to this latter side effect (an *Interest* message is not forwarded because it mistakenly generates a match) by implementing a *shared binary Bloom filter* in the control panel. If two Bloom filters work independently, the total false positive rate is the product of the two individual false positive rates of the two filters. By adding another filter after the first *Interest* message verification process, the rate of false positive can be significantly reduced. Every matching for an *Interest* packet raises another verification on the shared Bloom filter. If the second verification result is negative, this *Interest* will be forwarded downstream and added into the shared filter for a further check. If the result is positive, we can consider that this *Interest* packet is indeed a duplicate.

It is important to understand that this system based on two serialized Bloom filters differs from the implementation of a bigger Bloom Filter on each face. I emphasize here three reasons:

- Memory space. In order to have the same false positive rate, each PITi should be extended by the same size as the shared Bloom filter. Indeed, this extension mode should cost more memory than implementing only one shared Bloom filter.
- Performance. The *Interest* packets, which only need one verification at the PITi, will take a longer lookup time because a bigger Bloom filter has more hash functions than PITi.
- Relevance with regard to the internal process. A ContentName needs a second verification only if it generates a match in one PITi. The number of potential ContentNames to be tested for the second filter is smaller than the number of different ContentNames that arrive at the node. Only a fraction of all the received *Interest* messages requires two verifications.

For the same reason that I implemented a counting Bloom filter for each PITi in order to support the deleting, a mechanism to limit the false positive of the shared Bloom filter is also needed, since the returned *Data* packets do not remove the entries in the shared Bloom filter. I suggest to refresh the shared Bloom filter sometimes. I call it a RST mechanism. A RST notification is emitted from the control center, typically when the number of inserted elements reaches a threshold or on a regular basis. This RST mechanism does not damage the overall system behaviour, since the *Data* packet is forwarded only according to the "*bread crumbs*" left by the pending *Interest* messages in each PITi. The shared filter is not involved in *Data* object delivering. In DiPIT system, the repeated *Interest* packet is dropped only when both of the two level filters give a positive verification, thus the only side effect of the RST mechanism is to forward one more time the duplicated *Interest* packet and waste some networking bandwidth.

3.5.3 Main DiPIT algorithms

I present the pseudo-code of DiPIT algorithms at the reception of both *Interest* packet and *Data* packet in Algorithm 1 and Algorithm 2, respectively.

Incoming *Interest* An *Interest* packet that arrives on a given face is firstly checked in Content Store. If the Content Store does not have any matching *Data* object, the *Interest* packet is then checked in the PITi associated with this face. Here has several cases: (1) a negative result means that the *Interest* message never came in. It is forwarded to the FIB and its footprint is added in the filter; (2) a positive result means that either the *Interest* packet has already come, or it is a false positive. It is then checked in the secondary shared Bloom filter; (2a) if the second filter gives a negative answer, the *Interest* message is forwarded to the FIB and its footprint is added in the second filter; (2b) on the contrary, a positive result means that this *Interest* message is a duplicated emission, the CCN node blocks it.

Algorithm 1: Treatment of *Interest* packet in DiPIT

```
Input:
  interest: incoming Interest packet on face $i$
  $CBF_i$: the counting Bloom filter associated with face $i$
  $BF_S$: secondary shared Bloom filter
Main program:
 1: if interest matched in Content Store then
 2:    return matched Data
 3: else
 4:    if $CBF_i$ matching test on (interest) == false then
 5:       transfer interest to forwarding module
 6:       increase the counters of $CBF_i$ at the footprint positions
 7:       exit
 8:    else if $BF_S$ matching test on (interest) == false then
 9:       transfer interest to forwarding module
10:       add the footprint of interest in $BF_S$
11:       exit
12:    else
13:       block interest
```

Incoming *Data* The *Data* packet forwarding process is relatively simple. An incoming *Data* packet is verified in all the PITis, except the one where the *Data* packet comes. If a PITi contains a matching *Interest* message footprint, the *Data* packet is forwarded through this face and the footprint should be deleted from the PITi.

I do not describe a pseudo-code of the RST mechanism because there is no unique way to implement RST. I have proposed several implementations of RST, which can be roughly distinguished into two families: (*i*) the RST is triggered by an accumulated number of bit set to one, and (*ii*) the RST runs on a regular basis. The concrete threshold or the interval are depended on the size of the Bloom filters and the number of packets to treat, according to the CCN nodes.

3.5.4 Discussion on multiple same *Interest* messages filtering issue

In the design of CCN, if one node receives multiple *Interest* messages on the same content, only the first one is forwarded. The DiPIT architecture can avoid sending the duplicated *Interest* packet coming through the same face, but not for those coming from other faces, because every PITi performs independently from each other. This

Algorithm 2: Treatment of *Data* packet in DiPIT

```
Input:
   data: incoming Data packet
   i: face id
   Face[ ]: set of faces
   CBF_i: the counting Bloom filter associated with face i
Main program:
1: if (data) matched in Content Store then
2:    discard data
3:    return
4: else
5:    cache data in CS (optional)
6:    for all $i \in Face[\,]$ do
7:       if data matches $CBF_i$ then
8:          send data through face i
9:          decrease the counters of $CBF_i$ at the footprint positions
```

leads to an extra networking load. However, such event can only occur when two *Interest* packets arrive on distinct faces in a time frame of RTT (otherwise the second *Interest* packet would find the *Data* object in the Content Store). I analyse now this potential limitations. Let us suppose the worst case, *i.e.* several faces receiving the same *Interest* message within a *Data* message Round-Trip Time. No matter how many faces of the CCN node receive and forward the same *Interest* message, the impact on the whole network is only within one hop. The case is illustrated in Figure 3.12. The arrows denote *Interest* packet propagation. Node A receives the same *Interest* message several times from its different faces. According to its FIB table, all these *Interest* messages will be forwarded to node B. For node B, all these *Interest* packets come from the link between B and A via the same face. Thus only the first incoming *Interest* is further forwarded to the next hop C. The other *Interest* packets are blocked by the PITi tables because they arise a positive matching in the Bloom filter.

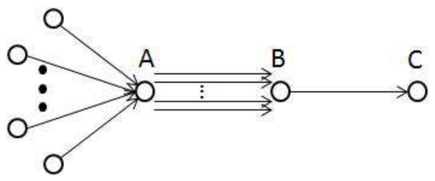


Figure 3.12: Duplicated *Interest* message coming through different interfaces

3.6 Evaluation

I performed several evaluations in order to validate that our DiPIT system can significantly reduce the required memory space for CCN PIT table.

3.6.1 Settings

I assume one networking line card holding 16 interfaces. The average *Interest* packet arrival rate ranges from 20Mpcks to 200Mpcks. Such a range is representative to several classes of routers. The *Data* packet RTT time is set as 80ms [PV11]. We used five hash functions for the counting Bloom filters. Each counter has 3bits, that counts until 8. The shared Bloom filter has a size of 1Mbits. In most previous works, the acceptable false positive probability is comprised between 0.01% and 10% [GWCL06, BM04, YFR09, QLC11]. The applications of today's IP network tend to tolerate a packet loss rate of 1% at least, I thus set our acceptable false positive in the range from 0.1% to 1%. Since the exact match is required in PIT table, the content identifier size is not important in our case. In the evaluations, the largest *Interest* packet arrival rate is 200Mpcks and the RTT time of *Data* packet is 80ms. According to *Little's Law*, the biggest number of elements in the PIT table is $(200 \cdot 10^6) \cdot (80 \cdot 10^{-3}) = 16 \cdot 10^6$, which can be represented by 2^{28}. Thus for the centralized hash table I used H-bit of 28bits. I also add 32bits H-bit in the evaluations, because 32bits is a common value in hash functions. As in [PV11], I assume 40Bytes *Interest* packet. The CCN name lengths are variable, so I take the middle value and give 128bits for each ContentName matching in hash table. For each hash table entry we also had to add 2Bytes to memorize the incoming interface identifiers.

3.6.2 Required memory size

First I evaluated the required memory space according to the average *Interest* packet arrival rate (λ). The DiPIT table size depends on the length of the Bloom filter and the number of faces n_{face}. The length of a Bloom filter m can be calculated from the acceptable false positive fp, the number of applied hash functions k, and the number of inserted elements n. From Equation (3.1), we have

$$m = \frac{-n \cdot k}{log(1 - fp^{\frac{1}{k}})}$$

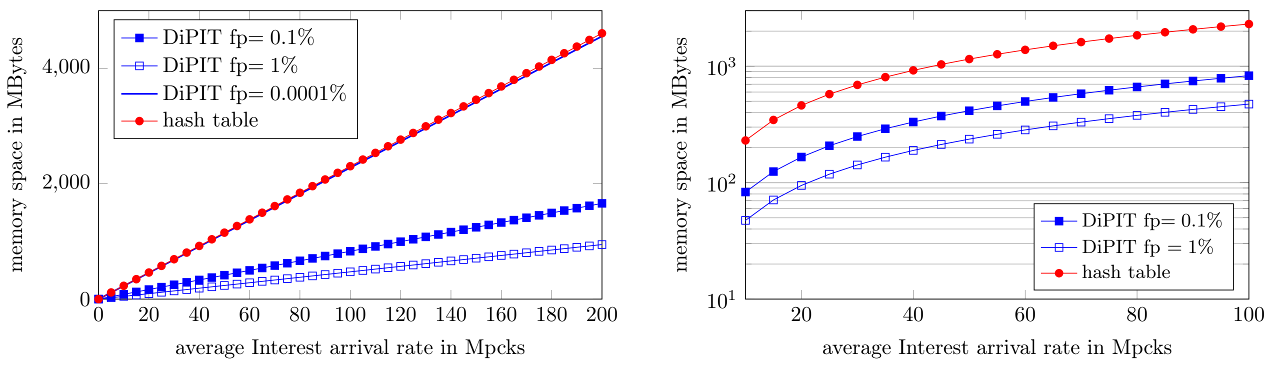
, where the n the product of the packet rate λ and the *Data* packet RTT.

Each required memory size for a counting Bloom filter is the product of the vector length and the counter size. We denote the counter size as $Counter$. Thus the estimated memory size of the entire DiPIT system is:

$$Size = Counter \cdot \sum_{i=1}^{n_{face}} \frac{-\lambda_i \cdot RTT \cdot k}{log(1 - fp^{-k})} \tag{3.6}$$

In Figure 3.13(a) (as well as in Figure 3.13(b), which is a zoom on the lowest packet arrival rates), I represent the entire required memory space in the condition of different acceptable false positive ratios of the DiPIT system. In this figure the number of hash functions is fixed as 5. Lines with squared marks represent DiPIT for two false positives values (0.1 and 1%). I also represent the size of the hash table for both in lines with circled marks. The DiPIT system based on Bloom filter clearly outperforms an equivalent system based on a hash table. Typically for λ_{in} = 100Mpcks, the required memory space for DiPIT with 0.1% of false positive is only 36% of the space required by the hash table.

Please note the represented sizes for DiPIT are for the *whole* DiPIT system. The required size for an individual PITi is only 51MBytes for λ_{in} = 100Mpcks. Therefore a PITi can be built on a fast memory. On the contrary, hash tables require an implementation on a large capacity memory, which has a longer access time. On the same space as the hash table, and for the same packet arrival rate, DiPIT would have a false positive ratio of nearly 0.0001% (plain line covered behind the hash table line).



(a) zoom out (b) zoom in

Figure 3.13: Required memory size in function of λ_{in}

In Figure 3.14, I studied more precisely the required memory space according to various false positive ratios. I fixed the *Interest* packet arrival rate at 100Mpcks (*Data* message response time is still 80ms). The 28-bits hash table needs 4.8GBytes. A 32-bits hash table can reduce the collision ratio, but it requires 77GBytes. In order to limit the false positive probability at 0.1% (corresponding to -3 in the x-axis), DiPIT needs around 1GBytes if the Bloom filters use three hash functions, but only 786MBytes and 690MBytes are necessary for five and seven hash functions, respectively. Thus we can also summarize that a router which does not deal with a high packet arrival rate can be implemented with more hash functions for saving the memory space. On the contrary, high level routers (*e.g.*, the edge routers at the peering point or the core routers) with a higher packet arrival rates would consider using less hash functions and relatively more memory to keep the acceptable performance speed.

Implementing relatively more hash functions can reduce the false positive ratio. However things are not that simple. The proper number of hash functions should be carefully chosen with regard of the number of packets which the CCN node face should treat and the implemented memory space. The Equation 3.2 shows the relation between the optimal number of hash functions, the vector size (the memory space) and the number of element (the Interest packets arriving rate). I represent in Figure 3.15 the probability to obtain false positives for two typical classes of memory, according to the average packet arrival rate. As can be shown, when the memory space is small (the m in Equation 3.2) and the Interest arriving rate is high (the n in Equation 3.2), more hash functions introduce a higher false positive rate. For example seven hash functions paradoxically lead to worse results than five or three hash functions on 32MBytes memory size at high packet arrival rate (see Figure 3.15(a)).

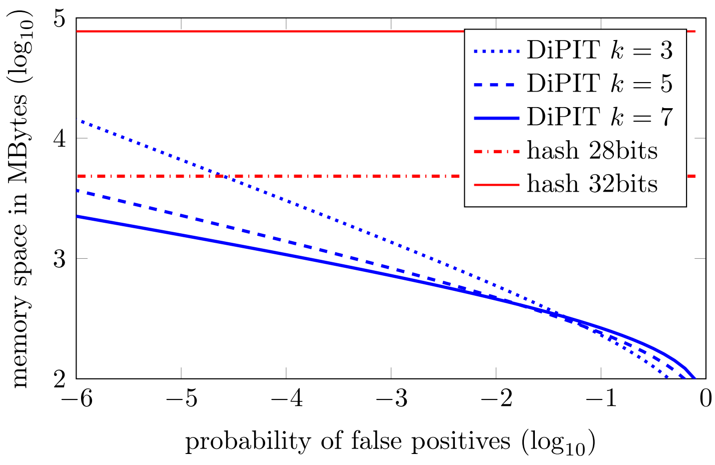


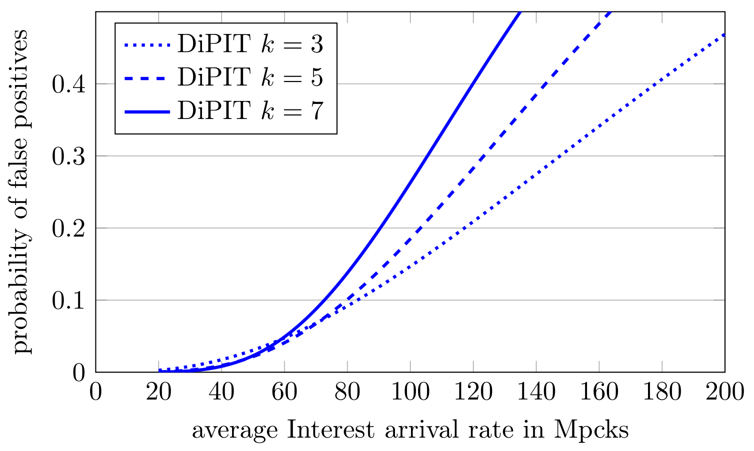
Figure 3.14: Required memory size vs. false positive

3.6.3 Bursty and multi-path traffic

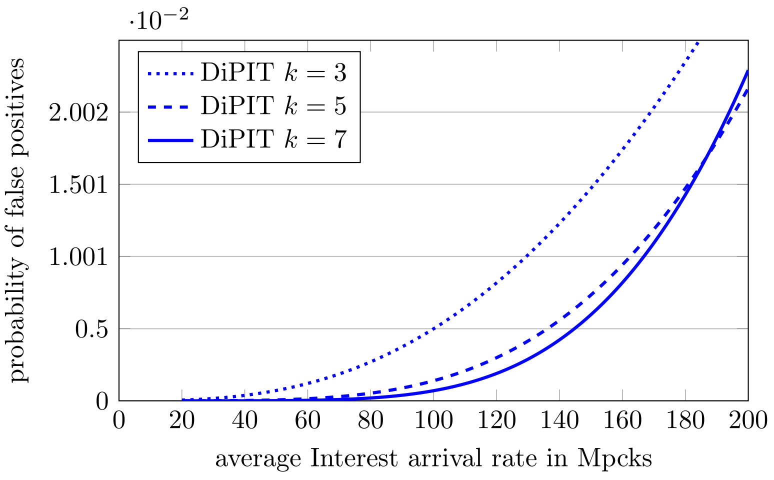
I now deal with more complex (and realistic) traffic. Firstly I consider that a given *Interest* message is received by several CCN faces of the same node. Then, I deal with fluctuant traffic.

If an *Interest* message comes several times from several different CCN faces (within a *Data* message response time), DiPIT is not able to filter the duplications because the same *Interest* messages coming to different faces are memorized independently at different PITis. A hash table is not impacted since it has only one centralized table. To be fair, I take into account this redundancy and I compute the actual space requirement for hash tables and for DiPIT according to the different probabilities that the received *Interest* messages from different faces are actually the same (Figure 3.16). For example, the probability of 1 means that the whole traffic carries only one same ConetentName. I show that only the 28-bits hash table can require less memory size than DiPIT and only when more than 80% of traffic is redundant. In reality, it is easy to imagine that the chance we have 80% of traffic about the same ContentName within one *Data* message response time (which is in order of millisecond) is quite small.

I now assume that the incoming traffic follows a *Poisson distribution* [KMFB04]. The more bursty the traffic is, the more false positives could be generated. In the meantime, it also means more packet losses for the centralized hash table. In Figure 3.17, DiPIT and the hash table are both designed to handle a traffic of λ_{in} = 100Mpcks. Each PITi has five hash functions. The left y-axis presents the probability of a bad event, which means either a false positive (in DiPIT) or a packet loss rate (in hash table). The right y-axis presents the probability of different packet arrival rates. I represent the Poisson law in red.



(a) memory size: 32 Mbytes



(b) memory size: 128 Mbytes

Figure 3.15: Probability of false positive vs. λ_{in}

A well-dimensioned hash table experienced no packet loss for traffic smaller than 100Mpcks, but the packet losses explode when the traffic is more intense. On the contrary, if we take a look at the curve slope, the false positive of DiPIT increases slowly. Typically, for 100Mpcks $< \lambda_{in} <$ 120Mpcks, which is a reasonable traffic burst, the probability of a bad event is still below 2% in DiPIT, although it reaches up to 15% for the hash table. Definitely, DiPIT is less sensitive to traffic burst.

3.6.4 Extra Data traffic load

Because a PITi produces a false positive, some *Data* packets might be sent out although no *Interest* message matches. Here I analyse the generated extra networking load. Let us suppose a CCN node with n_{face} interfaces receives nbr_{data}. I denote by fp_{j_i} the false positive ratio of the interface i at the node j. I set N_{hop} as the number of times the “false" *Data* packets can be forwarded. The sum of the “false"

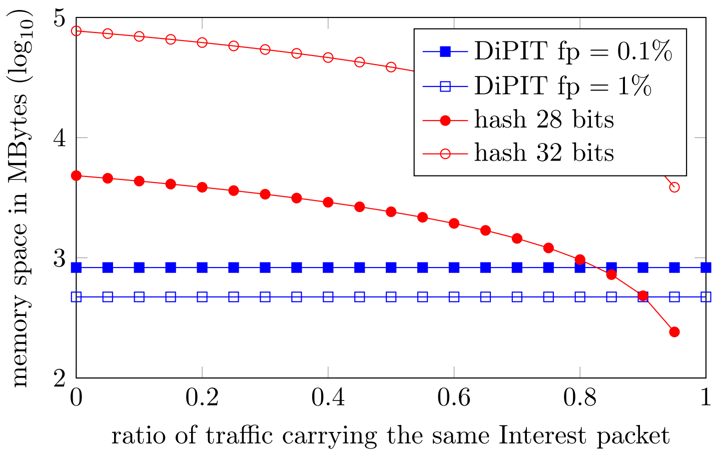


Figure 3.16: Memory size vs. ratio of traffic related to only one *Interest* packet

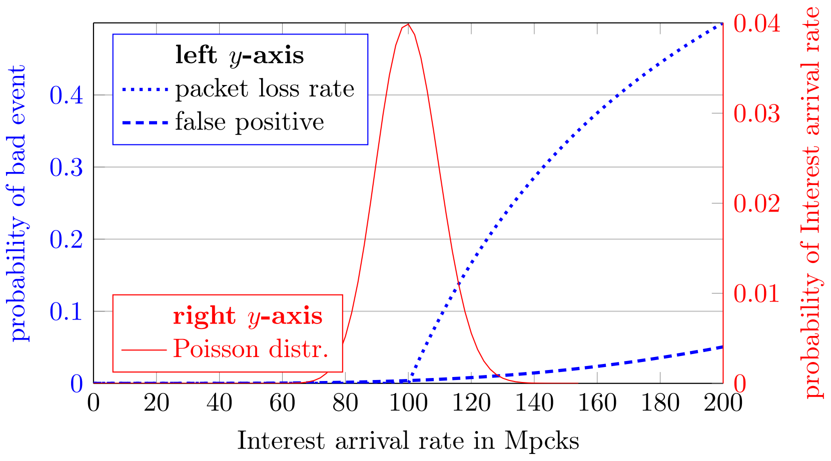


Figure 3.17: The burst vs. packet arrival rate following Poisson law

Data packets that are sent out by one node impacts the N_{hop} networking link such as:

$$Number_{data} = \sum_{j=1}^{N_{hop}} (nbr_{data} \cdot \sum_{1=1}^{n_{face}} fp_{j\_i}) \tag{3.7}$$

I fix the average *Data* packet arrival rate at 100Mpcks and the *Data* message RTT as 80ms. The acceptable false positive is 1%, 0.1% and 0.01%. From Figure 3.18 we can see that after the second hop, the extra traffic is nearly zero. In fact, one node can generate some extra *Data* packets but this extra traffic is stopped at the next hops because the probability several (even only 2) nodes produce the same mistake is very low.

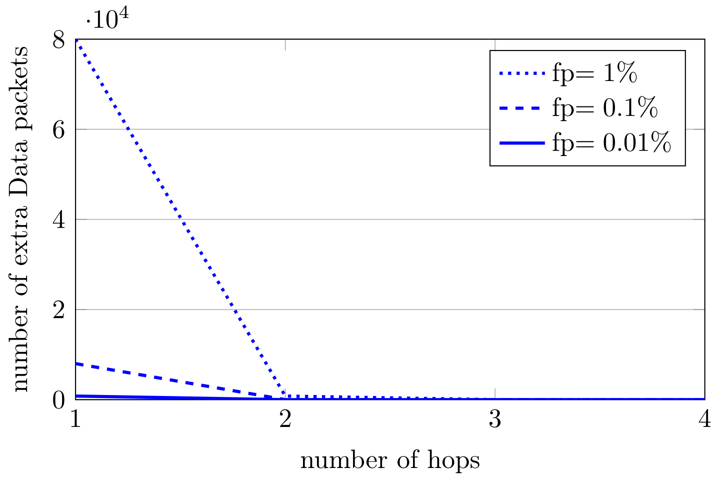


Figure 3.18: Extra *Data* packet load decreased during propagating

3.6.5 Discussion

These three evaluations prove that firstly the DiPIT can overcome the limitations of fast memory chips. When the packet arrival rate exceeds 130Mpcks, only the distributed system can allow SRAM technologies to be used for PIT. The trade-off is the false positive. DiPIT is especially the solution for high-level networking routers (like a core router or the peering router) that have a higher packet arrival rate. And since DiPIT requires less memory than a hash table, DiPIT can also be considered for CCN nodes with smaller packet rate arrival (if this node is not too sensitive to false positive effects). In a Bloom-filter based PIT system, more hash functions can reduce the filter size (Figure 3.14). If we seek a faster performance and we are less sensitive to the cost, we can use less hash functions (as $k = 3$ or 5) and larger filter in order to reach a targeted acceptable false positive. On the contrary, if we want to limit the cost, we can implement more hash functions (as $k = 7$ or 11) in order to reduce the memory space.

Secondly the DiPIT is less sensitive to the networking traffic burst, thus this system is more stable when the networking traffic environment changes, for example with flash crowd events.

At last, within the DiPIT system, the extra traffic which is introduced by the false positive can be limited with in known number of hops.

3.7 Implementation in CCNx

I also implemented the DiPIT system into the CCNx release (version 0.4.2), and performed some evaluations with a testbed at Telecom Bretagne. In my implementation, every CCN face of a CCN node associates with a counting Bloom filter. The global CCN node handle has a binary Bloom filter which is shared by all the CCN faces. This shared binary Bloom filter ia armed with a RST mechanism which is triggered by the number of inserted elements (the *Interest* names). The testbed is composed by several real machines.

3.7.1 Evaluation 1: in-line network

The first evaluation was performed with a linear topology which included 9 nodes, as shown in Figure 3.19.

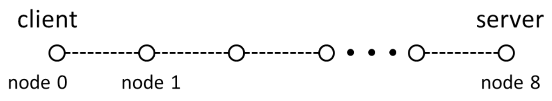


Figure 3.19: Linear topology of evaluation 1

Settings

- There was only one content server and one client.
- The client generated 10000 *Interest* messages. The ContentName popularity followed the zipf distribution with $\alpha = 0.7$.
- For each of the 9 CCN node, the PITi (counting Bloom filter) had $1Mbits$, the SBF (shared binary Bloom filter) was also $1Mbits$, and the threshold of the RST was set as 2.5%.

Results

- Among the 10000 *Interest* requests that generated by the client (node 0), there were 4827 different content names.
- The server (node 8) replied 4826 different contents.
- The DiPIT received 4821 *Interest* names and it "found" 82 repeated *Interests.* Thus we can calculate the false positive rate in the PITi is $\frac{82}{4821} = 1.7\%$.
- The DiPIT blocked 6 "repeated" *Interest* messages, which means the packet lost rate of the entire topology was $\frac{6+(4827-4826)}{4827} = 0.1\%$.

3.7.2 Evaluation 2: Geant networking

The second evaluation was run with a Geant networking topology, which was also composed by 9 CCN nodes, as shown in Figure 3.20.

Settings

- There was only one content server and one client.
- The client generated 10000 *Interest* messages. The ContentName popularity followed the zipf distribution with $\alpha = 0.7$.

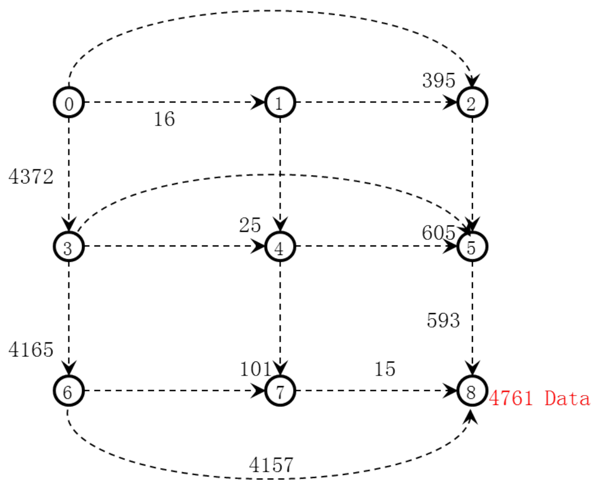


Figure 3.20: Linear topology of evaluation 2

- For each of the 9 CCN node, the PITi (counting Bloom filter) had $1Mbits$, the SBF (shared binary Bloom filter) was also $1Mbits$, and the threshold of the RST was set as 2.5%.

Results (the numbers near the dotted lines were the number of *Interest* messages)

- The Client (node 0) sent $4372 + 16 + 395 = 4783$ different *Interest* messages.
- The server (node 8) received $4157 + 15 + 593 = 4765$ different *Interest* requests.
- Thus the packet lost rate in total is $1 - \frac{4765}{4783} = 0.37\%$.
- The server (node 8) replied 4761 contents, thus approximatively the packet lost rate at one node is $\frac{4765-4761}{4765} = 0.08\%$, which matched the result the total packet lost rate and the result of the first evaluation.

3.7.3 Discussion

The results obtained from these two implementation evaluations do not concern the memory consumption, but the analysis of the false positive impacts. As we can see from the summarized results, the false positive ratio depends on the networking topology, but the highest ratio is still limited at less than 1.7%. And thanks to the two-step verification design, the Packet Loss Rate is limited at less than 0.1%. The DiPIT system (distribution of the PIT on each interface and the shared Bloom filter) then provides an acceptable limited ratio of false-positive and an acceptable solution for this point of view as well.

3.8 Case study: hierarchical network

In an operational deployment, a network operator chooses and deploys different types of routers for different locations based on different needs, different networking conditions or different budgets. Equally, the choice of a DiPIT system or the classic hash based one, and the choice of different memory chips should also be based on those aspects. I then analyse where the DiPIT-based CCN node can be more suited for deployment, having a case study on a hierarchical network (such as the Orange one). I especially identify the conditions under which a traditional hash table can be implemented and the results show that DiPIT can significantly expand the applicability of CCN in networks with today's memory technologies.

3.8.1 Analysis

A three level hierarchical topology is considered in this case, illustrated in Figure 3.21. The peering router is denoted as *root*. The *root* is connected with four core routers, which are denoted as I, II, III and IV. Each core router is connected to four edge routers, noted as 1, 2, 3, 4. The set of end-users which is connected to an edge router is noted as $A, B, \ldots, E$. Each set of end-users generates *Interest* packets with λ_{in}Mpcks. Each internal link has a $d$ms delay. The *root* has Dms delay with outside network. The acceptable false positive for the *root*, the core router and the edge routers are denoted fp_{root}, fp_I, and fp_1 respectively. The number of hash functions for the Bloom filter are k_{root}, k_I and k_1 respectively. The Content Store hit ratio for each level router is CS_{root}, CS_I and CS_1 respectively. The ratio of identical *Interest* packets among the traffic is $\tau_{interest}\%$.

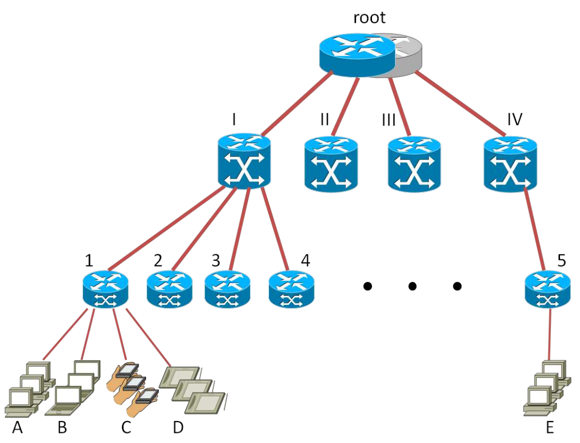


Figure 3.21: A hierarchical networking topology

The edge router 1 receives the *Interest* messages from four interfaces at a rate λ_{in}Mpcks. The RTT for each interface is $2 * d + D$ ms. I suppose that $\tau_{interest}$ are identical for the traffic of every interface and every router. Thus the overall distributed PIT size for an edge server is:

$$Size_1 = 2 \cdot 4 \cdot (1 - CS_1) \cdot \tau_{interest} \cdot \frac{-\lambda_{in} \cdot 10^6 \cdot (2 \cdot d + D) * 10^{-3} \cdot k_1}{log(1 - fp_1^{-k_1})} \tag{3.8}$$

The core router I receives the *Interest* packets from four edge routers, so it has a higher input packet rate (when the Content Store has a realistic hit ratio). The

Interest packet arrival rate at each interface is equal to the *Interest* packet leaving rate at the edge router up-streaming interface. Thus the overall distributed PIT size for the core router is:

$$Size_I = 2 \cdot 4 \cdot (1 - CS_1) \cdot (1 - CS_I)(\tau_{interest})^2 \cdot \frac{-\lambda_{in} \cdot 10^6 \cdot (d + D) \cdot 10^{-3} \cdot k_I}{log(1 - fp_I^{-k_I})} \tag{3.9}$$

Thus the overall DiPIT size for the *root* can be similarly derived as:

$$\begin{aligned} Size_{root} = 2 \cdot 4 \cdot (1 - CS_1) \cdot (1 - CS_I) \cdot (1 - CS_{root})(\tau_{interest})^3 \cdot \\ \frac{-\lambda_{in} \cdot 10^6 \cdot D \cdot 10^{-3} \cdot k_{root}}{log(1 - fp_{root}^{-k_{root}})} \end{aligned} \tag{3.10}$$

3.8.2 Settings

I now build the evaluation system. I consider that edge router receives *Interest* packets from four set of terminals at rate 10Mpcks and $\tau_{interest}\% = 95\%$. The transmission time on each internal link is 20ms, so RTT is smaller or equal to 80ms. I do not consider Content Store caching because our main motivation is to identify the right technologies for PIT, so the impact of $CS_{hit} = 0$ is identical on every technology. Since I showed that false positive has not a significant impact on the whole network, I set a relatively high acceptable false positive ratio for the root router (1%) while we use 0.1% for the edge and core routers. Finally, I used 7 hash functions for the edge routers, because the packet rate is not high. On the contrary, core routers and root routers need faster performance, thus I use 5 and 3 hash function for them, respectively.

3.8.3 Discussion

I now summarize how to choose the PIT architecture (traditional hash table or DiPIT) and the cheapest memory technology for the routers at different levels. See Figure 3.22. In short, all green blocks correspond to configurations where DiPIT allows an implementation of PIT table in today's memory technologies although it is impossible to make it with traditional hash tables (the blue blocks).

I now explain how to read Figure 3.22. I take the edge router as an example. If we can accept a false positive ratio that is larger than 0.01% (right column), DiPIT is always cheaper than centralized hash tables. And if the λ_{in} is smaller than 66Mpcks (calculated with $fp = 0.1\%$), it is better to implement with RLDRAM because it is cheaper. If we are sensitive to the false positive ($fp < 0.01\%$), the hash table is a better solution. However when the λ_{in} exceeds 86Mpcks, the hash table can no more be used. Then DiPIT with SRAM is the only option until $\lambda_{in} = 140$Mpcks. Recommendations for core and root router should be read the same way.

Figure 3.22 highlights the main contribution of this paper: I provide a way to let an operator make choices for the deployment of a CCN network. Consider for

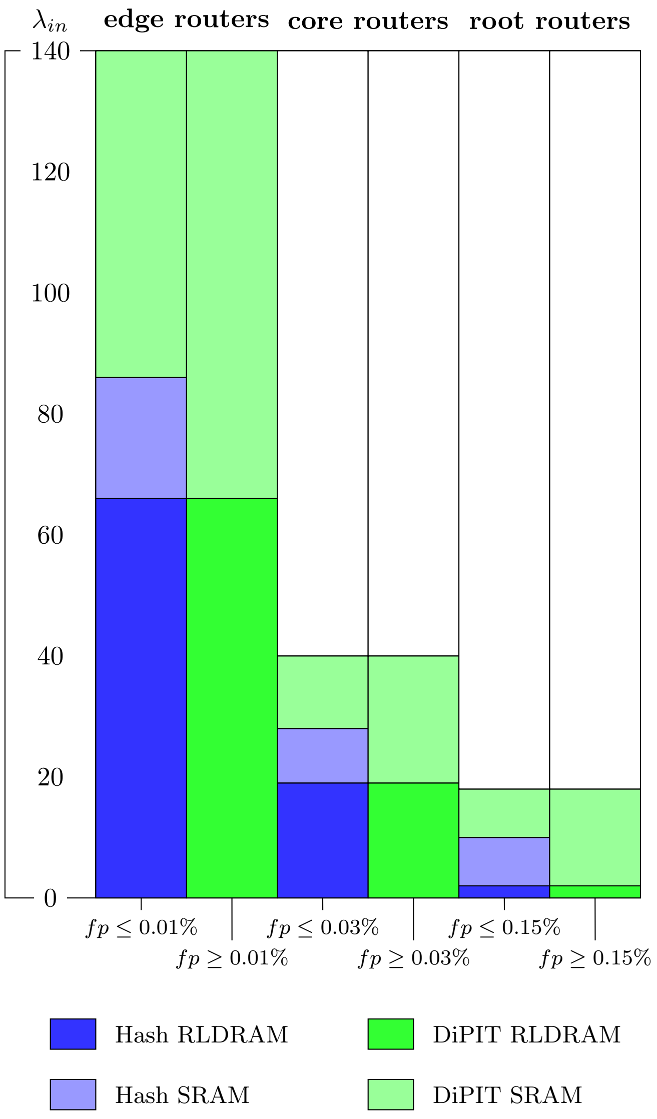


Figure 3.22: Summary of the best choices for PIT table, according to the cost

example an operator with incoming traffic at 15Mpcks, and the requirement to limit the false positive for the edge, core routers, and root router at 0.02%, 0.02%, and 0.2% respectively. The right choice for the edge routers is DiPIT+RLDRAM, for the core routers is Hash+RLDRAM and for the root router is DiPIT+SRAM.

3.9 Conclusion

Content-Centric Networking is a novel networking paradigm, which can bring many benefits for content delivery. If the basic principles are defined, improvements are necessary to reach a better performance and make CCN nodes more efficient. However today's memory technologies are not ready to support this innovation in term of memory space and access time, especially for the PIT table. In this chapter, I have proposed DiPIT, a distributed solution for the PIT component aiming at reducing the memory requirement in the CCN node. DiPIT is based on a PITi element, implemented on every CCN face with a counting Bloom filter, and a shared central element. Each PITi stores only the *Interest* packets coming on the associated face. The shared element enables to limit the false positive, inherent to the use of Bloom filters in the PITi. I analysed the feasibility and the scalability of the hash table. The hash table presents some limitations at the table size and the performance speed aspects but it does not produce any extra network load. The evaluations of DiPIT show that with a small acceptable false positive ratio, DiPIT can significantly reduce the memory space for implementing the CCN PIT table. Each technique has its advantage. The DiPIT system requires less memory space and enables implementations on fast memory chips, so it can handle higher packet arrival rate. However DiPIT introduces some extra network load because Bloom Filter can experience some false positive. I also showed that mixing different memory chip technologies is possible. Different networking conditions call for different approaches.

Chapter 4

A dynamical content-aware forwarding system for CCN

4.1 Introduction

In the previous chapter I discussed the impact of memory problem of the CCN PIT. The realistic implementation issue from the CCN impacts not only on the PIT table, but also on the forwarding system. In this chapter I focus on the CCN forwarding improvement. We all know already that the CCN protocol applies only two type of packets — the *Interest* and the *Data*, for content discovery and delivery. And it is noteworthy here that only the *Interest* packet is *routable* in the CCN transport layer. The forwarding information of *Interest* packets, which are based on names, are cached in the Forwarding Information Base (FIB) table of each CCN node. This table contains the prefixes of the ContentName and the outgoing faces information. The *Interest* packets are routed based on FIB according to the longest matching prefix lookups. The content providers broadcast the content advertisements into the network in order to announce the available contents. The reception of advertisement messages allows each CCN node to fulfill its FIB table.

In my consideration, current CCN forwarding design admits three critical weaknesses, which have the potentials to prevent a worldwide deployment of CCN. First a CCN human-readable domain name prefix does not enable efficient aggregation. In comparison, aggregation applied to an IP binary address is much more efficient. The consequence is an explosion of the number of FIB items. Second, with a world-wide scale network, the propagation of advertisements from content providers will generate an important networking traffic. In fact, the fundamental reason of these two points is that the potential number of content names is numerous, compared to the actual IP address number. At last, the on-path cache in CCN such as Content Store does not publish their cached objects. Since the CCN FIB is fulfilled by the content advertisements, that means the conventional CCN forwarding does not include the on-path caches as the content providers.

Thus, in this chapter I am trying to redesign the classic CCN forwarding system with the purpose of optimizing the CCN routing performance and of making it more suitable for a realistic implementation. Here I propose a content-aware dynamic forwarding design for CCN architecture, which contains three main parts that correspond the three points that I mentioned in the previous paragraph:

- *A content advertisement protocol.* In this design the advertisements are not propagated into the entire network, but only via certain paths with respect to the network topology and are collected by several selected nodes. Thus different CCN nodes in the entire topology do not have the same view of the entire available contents.
- *A new algorithm to fill the FIB.* This algorithm makes the fulfilling of CCN FIB is content-aware and dynamic. The FIB is not filled by the broadcast advertisements from the content providers, but rather from incoming *Data* packets in a dynamic manner. Hence FIB memorizes only the content name prefixes or the domain names that are locally popular.
- *A dynamic Interest downstream forwarding scheme.* The goal of this part is to make a better reuse of CCN *Data* packets that are cached on the delivery path. In current CCN implementations, Content Stores or other on-path caches do not announce the cached contents. That means the FIB based *Interest* message forwarding scheme does not include these in-networking caches. I propose here a new dynamic *Interest* message downstream forwarding scheme, which aims at discovering cached contents from downstream nodes.

Hereafter, I introduce firstly some related works about the CCN forwarding issue in Section 4.2. Then the details of content-aware dynamic forwarding system are presented in Section 4.3. Finally, I evaluate the feasibility of implementing our proposal in a CCN node in Section 4.4 before a short conclusion of this chapter.

4.2 Related Works

In Content-Centric Networking, or any other information-oriented networking solutions, the content-based forwarding and routing are the basic concept. Indeed, lots of previous work have focused on the CCN routing and forwarding issue, but most of them addressed on the content request forwarding [WZB13, SSRV11], or traffic congestion control [CGMP13, WRN$^+$13]. But less work addresses on how the contents get published or how the CCN FIB get filled. Indeed, there are various ICN solutions, that I presented in Chapter 2, and each solution has its own forwarding design. However those solutions are still based on either adding broadcast type messages, or including an additional element in the network [BCA$^+$12]. *NetInf* relies on a independent naming resolution system. *PSIRP* deploys the RendezVous nodes in its topology. These methods try to passively forward requests to the potential content locations without actually considering the name-based characteristics of data. At the CCN side, the authors from [WHY$^+$12, HAA$^+$13] aim to extend the IP OSPF protocol for the CCN networking — the OSPFN (details are presented in Chapter 2). But I am afraid this proposal is not CCN-friendly for three reasons. First, each content, or at least each content prefix will create an OLSA message for the announcement, and the OLSA messages are propagated through the entire network then reaches every CCN node. Thus such implementation will generate a lot of OLSA traffic into the whole network. Second, each node has the same and entire catalog of all the available contents in the whole networking topology. This design matches the needs of IP networks but it is inappropriate for CCN, since CCN nodes look for contents only when they receive requests for them. Indeed some contents are only popular at certain geography regions [WPD+10, VdMSK02, BSAS13], and different regions do not have the same catalog of popular contents [GHM12]. This calls for CCN nodes having a local view of contents availability. At last an OLSA message still contains the *router-id* information and a Shortest Path calculation is still performed based on the *router-id*, that means the OSPFN is still a location-based broadcast algorithm, but not a real content-centric one. These three points motivates the exploration of new content-aware forwarding solutions—the content-aware dynamic CCN forwarding.

4.3 A Content-Aware CCN forwarding structure

In this section I present the dynamic content-aware forwarding solution for CCN. The proposal requires a new architecture for the "forwarding unit" of CCN node. The new structure is given in Figure 4.1. In the new system, I add a new element, which is called Dynamic Interest Forwarding Table (DIFT). We also significantly change the way the FIB is filled. To avoid confusion between today's FIB implementation and our proposal, we say hereafter "content-aware FIB". The fundamental idea behind this proposal is that I do not use broadcasted advertisement messages to decide the forwarding. Instead, I leverage Data delivery paths, which allow each CCN node to take into account the local popularity and availability of data. The Content-Aware CCN forwarding structure contains three main compositions, the details of each are presented one by one in the following subsections.

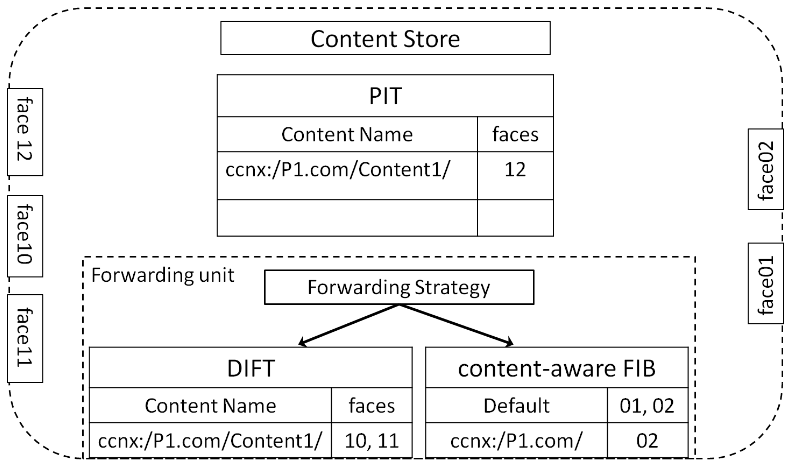


Figure 4.1: The node structure of the content-aware forwarding design

4.3.1 Content Advertisement Protocol

I aim to realize a content-aware forwarding solution but it does not mean that I do not need any content advertisement mechanism at all. Content providers still need to advertise content, otherwise the network will not be aware of what providers offer. However, in my proposal, these advertisement are not broadcasted in the entire network. Furthermore, CCN node does not rely on these advertisements to fill content-aware FIB. In the following, I use indifferently content advertisement and content publication, which is a common term in network.

The *default face* is an important concept in the content advertisement protocol. The default face(s) at each CCN node are the interfaces that are connected to the "upper" node, that is the next routers on the path to the peering nodes of an Autonomous System. It can typically be the root nodes of a hierarchical networking topology or the border routers of a mesh networking topology.

The configuration of the default faces can be realized either manually, or through the Shortest Path algorithm. For example in a hierarchical networking topology, the network operator knows how nodes are connected with the upper node(s), the neighbour nodes. Thus Internet Service Provider (ISP) can easily configure the default face(s). In a mesh topology, the default face(s) can also be configured so that the shortest path to the border nodes are default faces.

Let say a content provider wants to advertise a new content C. The equipment that is in charge of emitting content advertisement is the first router of the content provider delivery infrastructure. The first router first adds the advertisement information into its FIB. Then it propagates the publication through all its *default faces.* The next nodes receive this publication, and perform the same process, *i.e.*, add the publication into their FIBs and forward it through the default faces. Finally the publication reaches the border nodes of the Autonomous System of content provider. Thus all the node that are on the publication forwarding path and the border nodes have the published content C in their FIB. The border nodes also collect all content publication of other connected ASes.

Network operators can decide the number of border nodes in their AS, according to network topology and size. Overall, this proposal matches a fundamental principle of Internet, which is to let each network operator manage its own network. Here, the diffusion of content advertisements actually depend on the policy of each network operator. Other ways to collect and exchange content include distributed implementation based on Distributed Hash Tables (DHTs).

4.3.2 Content-Aware Dynamic FIB

To fill a content-aware FIB, many options are possible. This proposal is to leverage incoming Data messages. When a CCN node receives a Data message, it memorizes the domain name of the ContentName and the incoming face in content-aware FIB. It can thus forward the subsequent *Interest* packets for the same domain name through the face from which it received the Data to reach the content container.

The algorithm of the incoming Data at the content-aware forwarding structure is shown in Figure 4.2. The content-aware FIB does initially not contain any forwarding information, except the default face(s) configuration. Once the node receives a content

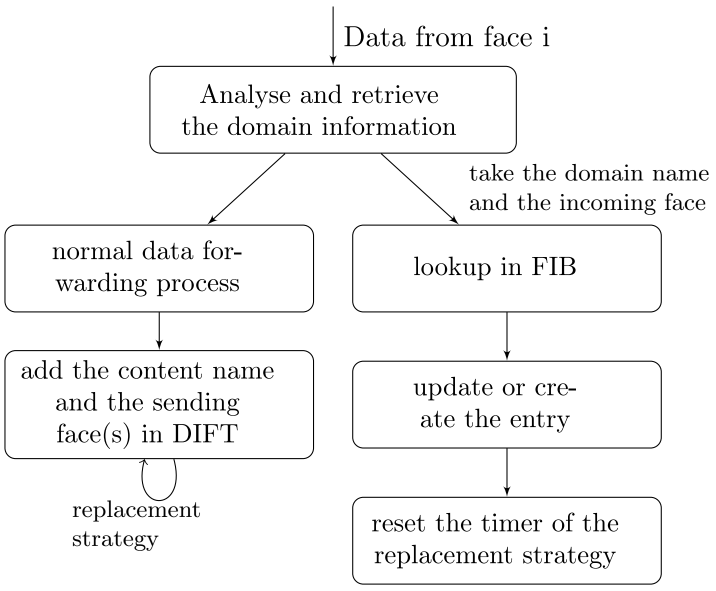

Data from face i
Analyse and retrieve the domain information
take the domain name and the incoming face
normal data forwarding process
lookup in FIB
add the content name and the sending face(s) in DIFT
update or create the entry
replacement strategy
reset the timer of the replacement strategy

Figure 4.2: Algorithm for the processing of incoming Data packet

packet, it retrieves the domain name and adds it into FIB together with the incoming face information. If the content-aware FIB does not have any entry on this domain name, it creates one. If there is already an existing entry on this domain name, it updates the entry with the new arrival face. It is allowed to have several outgoing faces. Typically contents from a given domain can be provided by different providers, for example the original server or a CDN repository.

Each entry is thus associated with a face list, which is updated at reception of every matching Data packet. I opt for a simple First-In-First-Out (FIFO) replacing strategy to avoid too long face lists. If one entry has too many faces in the list, the older faces should be replaced with the newer ones, so that memory space is saved and expired information are discarded. For the same reason, the entire content-aware FIB also includes a replacement strategy (LRU or Random) for the entries (different domain names).

When the node receives an *Interest* request, if this *Interest* packet matches the content-aware FIB on the *Interest* domain name, this *Interest* message is forwarded through the faces of the matching entry. If the domain name is unknown by FIB, the node sends them through the default interface(s).

4.3.3 Dynamic Interest Forwarding Table

In addition to the above algorithm for content-aware FIB, I would like to make a better use of Content Store (CS) in routers. CCN Data are cached along the propagating path, and each node has a local cache. However CSs do not advertise content in current CCN protocol. Thus the goal is to overcome these limitations and to exploit cached content in the downstream CSs, which can become content sources.

I propose a new table, which is called *Dynamic Interest Forwarding Table* (DIFT), to take advantage of the downstream CSs. A DIFT stores recent Data packets that have been forwarded to downstream nodes. The idea is that, since downstream nodes have generally to manage less traffic, it is possible that a request for a content that has been recently treated by a node is still available in the CS of one downstream node. In other words, contents cached in CS are expected to stay longer in downstream routers, so routing *Interest* messages to them makes sense.

The structure of DIFT is similar as the PIT. Each DIFT entry contains two columns, one is for the ContentNames and the other one is for the related face identifier(s). When a Data packet is returned to a CCN node, a PIT lookup process is performed. The Data message is forwarded through the faces of the matching PIT entry, and the matching PIT entry is then deleted. DIFT table is used to temporarily memorize the deleted PIT information. It may serve in a very short term to forwarding *Interest* messages (as shown in Figure 4.2).

I give in Figure 4.3 the flowchart for *Interest* packet processing in the content-aware forwarding algorithm. When an *Interest* packet arrives at a CCN node (and requested content is not stored in the local CS), after the PIT process, there are two forwarding steps: DIFT lookup and content-aware FIB lookup. If a matching entry is found in DIFT, it means that one copy of the requested content has been recently sent to some downstream nodes, so it might still be stored in their CS. Hence, the *Interest* packet is forwarded to those downstream nodes. In the meantime, since the content might not still be in the CS of downstream nodes, the CCN node also forwards *Interest* message towards a reliable content provider through the default face of content-aware FIB.

How to use the DIFT table and the content-aware FIB can vary and depend on a forwarding strategy based on the concrete networking conditions. When a node receives an *Interest* message, it can choose whether to consult the DIFT or not. If so, according to the different forwarding information in both FIB and DIFT, the node can decide to send the *Interest* message in either parallel or serially. That is the role of the strategy bloc I add between the FIB and DIFT in Figure 4.3.

The DIFT table is also implemented with a purge timer, because the behaviour of DIFT is far more dynamic than content-aware FIB. First, it memorizes longer name prefixes while FIB stores only domain names. Second, the forwarding information in the DIFT is related to the dynamic content container (*e.g.* the CS of the downstream CCN nodes or the end-users of a P2P system), while the content-aware FIB is related to the relatively reliable content providers (*e.g.* the content provider, the CDN repositories or the ISP caches). Since the content sources that are included in DIFT are not as reliable as those in content-aware FIB, the entries at the DIFT should be updated and refreshed faster with a simple replacement strategies (for example the FIFO or Random policies).

4.4 Evaluations

I performed some evaluations of our proposal to study the feasibility of our content-aware CCN forwarding scheme, with a focus on memory in CCN nodes and on the response time. The motivation is that previous works on CCN nodes have revealed

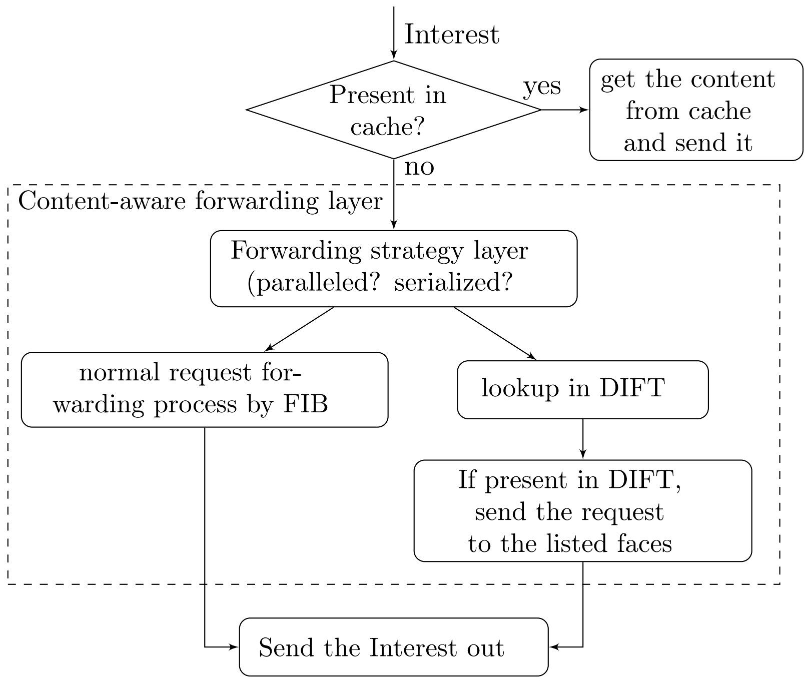

Interest
Present in cache?
yes
get the content from cache and send it
no
Content-aware forwarding layer
Forwarding strategy layer (paralleled? serialized?
normal request forwarding process by FIB
lookup in DIFT
If present in DIFT, send the request to the listed faces
Send the Interest out

Figure 4.3: Algorithm for the processinf og incoming *Interest* packet

that memory issues within CCN node can seriously prevent the implementation of some smart but not realistic algorithms, *e.g.* [YMT+12, PV11]. I thus decided to have a comprehensive look at memory implementation issues.

4.4.1 Settings

I consider a typical CCN node, which is deployed in a network where in average each node has 3 external links (*e.g.* the Abilene topology). Moreover I consider one *default face*. I suppose there are 12,000 content that are advertised in the network, including 2,000 which have been received by this peculiar CCN node. The node receives in average 200 *Interest* packets per second and each *Interest* packet experiences a 10 ms *Data* object RTT. The size of each entry size in the traditional FIB and in the DIFT are 32 Bytes (30 Bytes for the ContentNames plus 2 Bytes of the face identifiers), and the average entry size of content-aware FIB is 22 Bytes (in other words the domain name is in average 22 Bytes long).

The popularity of the set of contents is a critical parameter. I assume a Zipf popularity distribution with a parameter α. Previous works have revealed how critical is this parameter [RR12a]. In the following, we make this parameter vary in typical ranges from 0 to 2.

4.4.2 Impact of Content Popularity

In Figure 4.4, I present the memory requirement according to the parameter α of the Zipf distribution. In the traditional CCN publication algorithm, the CCN node receives all content advertisement, and FIB is filled accordingly. Hence the size of classic FIB is independent of α. The content-aware FIB is of course smaller. I evaluate this in Figure 4.4.

I first analyse the solution with only content-aware FIB. The size of this content-aware FIB is the highest for small values of α. The more uniform is the content popularity, the higher is the probability that the CCN node has to forward a *Data* packet related to every content in the catalog, and consequently the larger is FIB size. When α increases, some contents are never requested, so this CCN node ignores these contents.

I now study the association of content-aware FIB and DIFT. I assume every DIFT entry stays 10,000 time *Data* RTT. As can be expected, the size of DIFT is more sensitive to content popularity. The overall memory requirement is close to the one required by the traditional FIB implementation when α is around 0.8, which corresponds to the generic web traffic [FRRS12]. For values of α bigger than 0.8 (*e.g.* it is case fo VoD traffic [FRRS12]), the overall memory requirement is smaller than the traditional FIB.

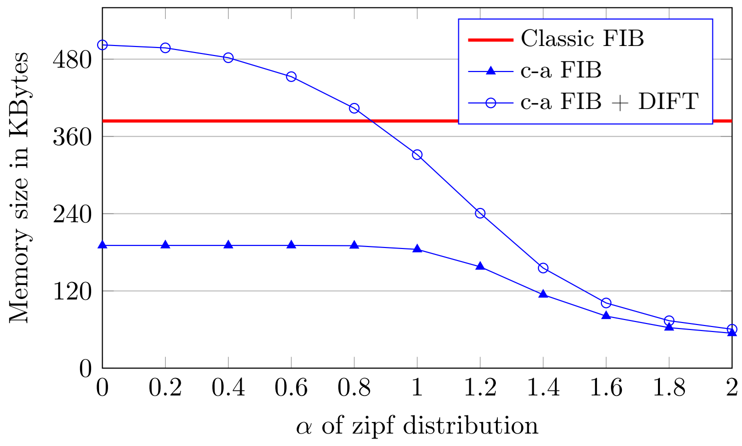


Figure 4.4: Memory requirement vs. content popularity

4.4.3 Impact of Network Topology

I now explore the impact of network topology on memory requirements. In Figure 4.5, I represent FIB size according to the degree of CCN node. Different curves represent different values of α for content popularities. It has to be recalled that *Data* packets coming from the default face are not memorized. A node with many connections to other CCN nodes should thus store more *Data* packets because the overall traffic coming from non-default face is bigger. In current core networks, each core node is in average connected to 3-4 links (*e.g.* Abilene topology or Géant topology). In this case, our content-aware FIB consumes not more than 50% of the classic FIB memory.

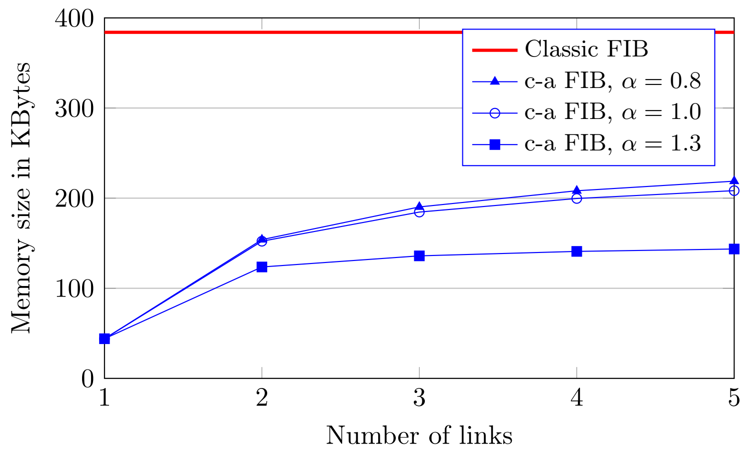


Figure 4.5: FIB memory requirement vs. the number of links

4.4.4 Impact on Entry Storage Duration in FIB

FIB incorporates a timer. Entries are automatically removed from FIB when they are too old. In Figure 4.6 we analyze FIB size according to the entry saving duration. For small values of α, the table size reaches it maximum even with a shorter saving duration, that means the FIB is filled fast. On the contrary, for example when $\alpha = 1.3$, even if entries are kept for 30 minutes, FIB does not reach its maximum value. This is can be explained by observing that FIB is filled much slower when content popularity is less uniform.

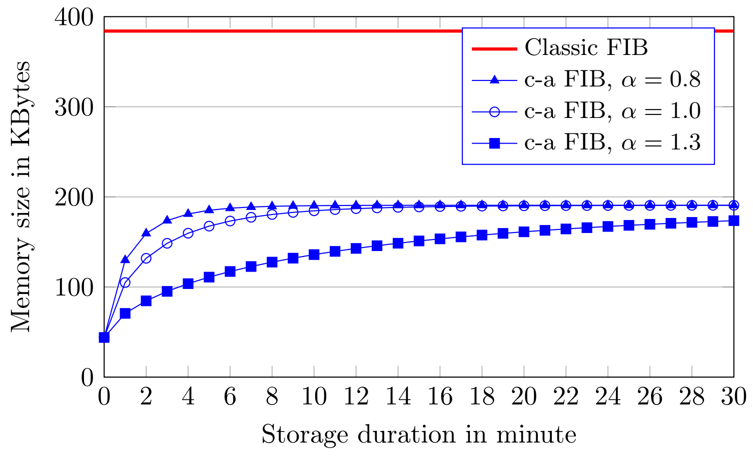
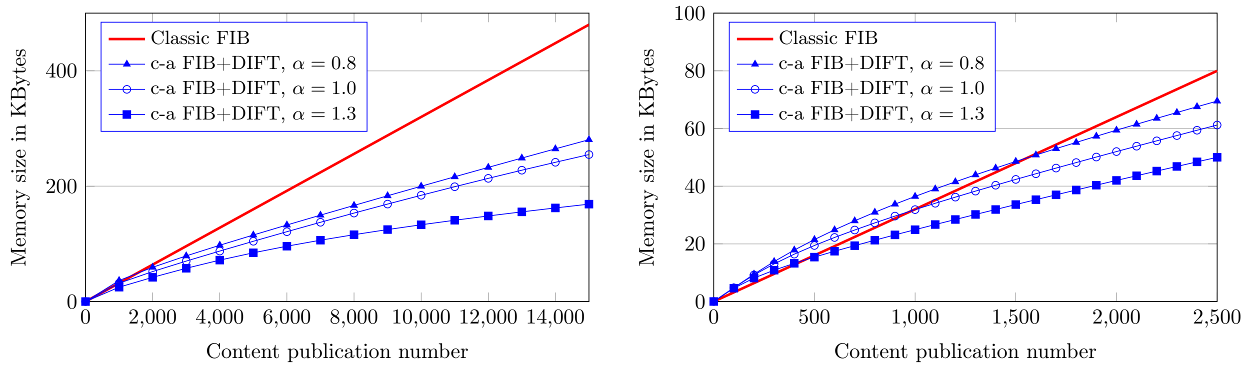


Figure 4.6: FIB memory requirement vs. the FIB entry storage duration

4.4.5 Impact of Catalog Size

In the previous evaluations, the size of catalog (*i.e.* the number of different contents is set to 12,000. In Figure 4.7 I analyse the impact of the number of publications on the memory size requirement. Intuitively, more publications means more memory space needs. We can see the size of classic FIB linearly increases with the number of publications. In Figure 4.7(a), I show a wide evaluation with potentially very large catalog. Here, the content-aware FIB enable a significant gain in terms of memory requirements. In Figure 4.7(b), I emphasize the special case of very few different contents. Here, the traditional FIB can be more efficient than our content-aware FIB only for $\alpha = 0.8$. Otherwise, content-aware solution is always at least close to the traditional FIB, or far better.



(a) zoom out (b) zoom in

Figure 4.7: Total memory requirement vs. the number of publications

4.4.6 Impact of DIFT hit-ratio

The Content Stores of downstream nodes have their own replacing strategies. Of course, all the *Interest* messages that are sent by the upstream DIFT table do not generate a match. In this evaluation I suppose the downstream nodes can offer a 75% CS hit-ratio. Thus I define the DIFT hit-ratio as the number of incoming *Interest* packets that gets a DIFT match and get the downstream CS hit, over the number of all incoming *Interest* packets. In Figure 4.8 we see that applying a bigger DIFT size can offer a better DIFT hit ratio. And a bigger Zipf α can raise also a better DIFT hit-ratio. For example the Zipf distribution of the generic web traffic has an $\alpha = 0.8$, the 64 KBytes, 128 KBytes and 256 Kbytes DIFT can offer a 49%, 58% and 68% DIFT hit-ratio, respectively.

4.4.7 Impact of response time

Since according to the DIFT table, some requests for the popular content are forwarded downstream closer P2P users, the content delivery time can be improved. In this evaluation I analyse with the content-aware forwarding scheme, the average *Data* packet response time in function of the content popularity. I suppose here a hierarchical networking topology which has three levels: the root nodes, the core nodes and the edge nodes. The average round-trip times are 10ms for the links between the connected levels, and 80ms for attaining the original servers. For example if the

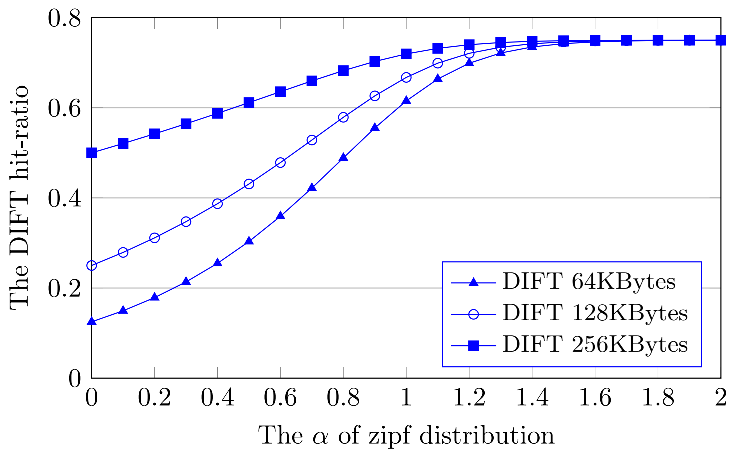


Figure 4.8: DIFT hit ratio vs. the content popularity distributions

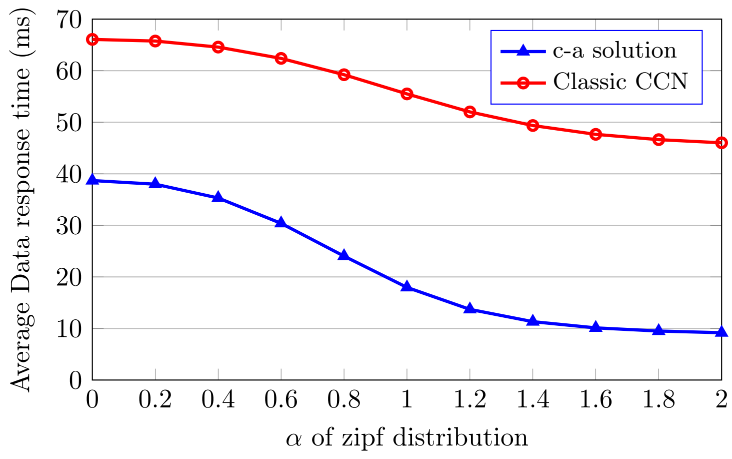


Figure 4.9: Average *Data* response time vs. the content popularity distributions

content-aware DIFT can find a matched content at the P2P end users which are connected under the same edge node, core node or root node as the demander, the *Data* packet delivery time is 10ms, 20ms or 30ms, respectively. The Content Store of each node can offer 0.3%, 0.2% and 0.1% content hit-ratio at edge node, core node and root node, respectively, since down level nodes receive less intensive traffic than high nodes. If the content request gets satisfied at the Content Store of the node of a certain level, the delay is thus the half of the related *Data* message Round-Trip Time. From Figure 4.9 we can see that by implementing the content-aware forwarding solution, *Interest* message experiences a shorter response time for whichever content popularity. We can also observe that the difference between content-aware solution and the classic CCN forwarding for a bigger α is greater than the one for a smaller α. This can be explained as the more popular a content is, the more possible that the *Interest* packets are served at the downstream nodes, which offer a shorter response time.

4.4.8 Discussion

The above evaluations valid that the content-aware forwarding design is more efficient in terms of the memory implementation. In various networking environment, for example different content popularities, different networking topologies, different FIB entry storage durations, different content catalog size, different caching hit-ratios and different content response times, it can significantly reduce the memory requirement compared to the classic forwarding system. Only when the content popularity is more uniform (*e.g.* the α parameter of Zipf distribution is less than 0.8), or if the content catalog size is small (*e.g.* the total number of available content is not more than 1000), the content-aware forwarding system needs more memory. But this extreme condition is rare in the normal networking environment. Even if this case happens, the additional memory consumption is less than 30% of the required classic CCN forwarding table needs.

4.5 Conclusion

I present in this chapter a content-aware dynamic forwarding proposal for CCN. In comparison to the traditional broadcast-based approach, this approach is more respectful of the information-centric principles. The content-aware forwarding proposal let each CCN node only store domain names that are close to it, thus FIB information differs among CCN nodes to reflect localized variation of popularity. In this approach, we can also make a better use of in-networking caches in the downstream nodes. The evaluations demonstrate that such CCN node implementation is feasible with regard to memory requirements, with table size that is not larger than in the traditional approach.

Chapter 5

Interconnecting CDN service with CCN networking

5.1 Introduction

The Internet is ineluctably becoming a gigantic content delivery platform [eri12]. To enable this shift, Content Delivery Networks (CDNs) deploy massive content *repositories* as close as possible to end-users. A CDN repository is authorized to serve contents on behalf of content providers, *i.e.* to intercept requests for addresses in the domain names of their clients (content providers) and to redirect these requests toward their repositories. Since a repository can fulfil a large proportion of requests for addresses in the domain names of its clients, it should be considered as a natural destination for the said domain names. Meanwhile, the Content-Centric Network, as well as other ICN solutions, are suggested for a radical shift toward information-centric approaches where content discovery and delivery are directly implemented into the routing protocol with on-path caches [ADI$^+$12]. However these proposals have ignored repositories so far.

With regard to the flourishing literature related to the management of caches in CCN, I understand that many network scientists believe that the CCN-based *universal cache*, which leverages storage facilities embedded, can totally replace the independent CDN repository. But this way of thinking means that the hybrid CCN/CDN node should be able to ensure high capacity of routing and forwarding functions, as well as efficient content storage and management. I think there is a misunderstanding about what routers will be able to offer. Networks of caches have been extensively studied a decade ago until it was revealed that cooperative caching has inherent weaknesses [WVS$^+$99]. In particular, the *filter effect* makes that the traffic after going through one cache does not have any more the characteristics that enable good cache performances [Wil02]. More recent studies have shown that multi-path request forwarding does not help in preventing the filter effect [RR12b]. In practice, protocols for wide network of caches (*e.g.* HTCP) are still unused. Instead, practical implementations of cooperative caching strategies (*e.g.* squid) are deployed in front of servers rather than in the network.

I follow the initial vision from [JST+09] where Content Stores (CS), which are low-volume fast-access memory technologies in CCN nodes, are confined to improve *Data* packets retransmission in case of packet loss, rather than a permanent content container. I believe in the existence of CDN repositories, which implements complex strategies to optimally utilize a high volume storage. These repositories can be controlled by a CDN provider, or a network operator or even a content provider like it is the case with Netflix [Ope12]. These CDN repositories are also very different from routers since they are optimized for fast delivery of content and not for traffic management. In other words, a repository is not a CCN node with huge volume of storage data, it is a server, which focuses on replying to requests on behalf of a small number of content providers. In this vision, the techniques that have been proposed for discovering contents in network of caches (*e.g.*, [SNS+12]) are not relevant. Besides the technical issue, from the business point of view, each actor (network operators and CDN providers) would not like to break their current business models.

The motivation of this chapter is to make both CDN repositories and CCN coexist. This challenge is fundamental with regards to the key role of CDN repositories in content delivery and to the importance of name-oriented routing in the future of networks. Thinking of the coexistence, I identified here two major questions. The first one is **how do CCN nodes route *Interest* requests to CDN repositories?** In CCN, the content servers must advertise the contents that they offer. The CCN FIB is filled with such content advertisements. Thus the CCN nodes are aware of the available contents and where to forward the *Interest* packets to. Then one quick idea is that we can ask the CDN repositories announce too what they have cached by using the any content publication protocol (as I presented in Chapter 4). The CCN nodes which receive these CDN repository advertisements add the information in their FIBs. It can work with a small number of contents but it can not be efficient in case of billions and billions of different contents. If every content is advertised with its full name, it will introduce a big network overhead and a huge FIB table size. Since the CDN service is for several certain content providers, the contents from the same domain names can be aggregated to a smaller identification name. Thus all the *Interest* messages carrying the same domain names are successful forwarded to the CDN repositories. For example, instead of advertising the full name of two contents (ccn.Orange.com/Video/Lannion/Demo1 and ccn.Orange.com/VoD/film1) that both will be registered in the CCN node, we can assume that only one aggregated name (ccn.Orange.com/) can be advertised by the CDN nodes. It will largely reduce the necessary memory size in the CCN routers and ease the forwarding process as well, since it will be faster to retrieve information. However, the CDN repositories hold only the most popular contents offered by the service providers, not the full catalog. So Here comes the second question. **How do CCN nodes get content from the original servers in case of a CDN miss?** CCN protocol differs from IP protocol as it does not support negative reply or address-based redirection. If we use aggregated information in the FIB as mentioned earlier (ccn.Orange.com/), all the *Interest* messages related to this domain name will be sent towards the CDN repositories. But in case the target CDN repository does not have the contents, the preceding CCN node cannot be informed and the end-user will never get the content, whereas it is available in the original server.

In order to realize the cooperation between CCN and the CDN caches, I propose here a novel CCN node, namely *cRouter*, which aims at integrating CDN repositories in CCN. The main feature of a cRouter is that a CDN repository can "plug" on it. From a business perspective, a cRouter is managed by an Internet Service Providers (ISP) while the repositories are not necessarily. An ISP operator can deploy cRouters on-the-fly in its CCN network without affecting other nodes. Since the ISP operator is free to deploy any number of cRouters at strategic locations in its network, the proposal preserves one of the most fundamental principles of the Internet, which is to let the ISP operators actually operate their networks by themselves. Furthermore, this proposal does not require changing CCN, since I do neither modify any CCN message, nor create any new message.

5.2 Background and motivation

In this Section, I first give a shot presentation of some additional business-oriented motivations for the deployment of repositories in networks. Then, I show that naive implementations with current CCN nodes fail.

5.2.1 Business Considerations – Analyzing the CDN Revolution

Several theoretical models have been proposed to capture the reality of exchanges between business entities in the context of massive content delivery [MCL$^+$11, DD11]. On this side, I do not focus on precisely modeling monetary interactions, but rather on understanding the motivations behind the strategies of today's stakeholders. There are four main actors are distinguished:

1. **Content providers** want to serve their subscribers (residential end-users) without having to engage specific deals with ISPs. They also want to maintain exclusive relationships with their clients (monetizable service personalizations), thus they want to be notified of each action of their clients. Therefore traffic interception by un-authorized third-parties is not acceptable.

2. **CDN providers** have deployed a content delivery infrastructure (servers, possibly backbone networks) and have agreements with content providers. They want to get rid of variable costs and to find added value in content handling. CDN providers also want to participate to global infrastructure investments, in order to better balance and value their core assets (technologies, knowledge and current infrastructures).

3. **ISPs** receive money from residential end-users. They want to maintain cost structures under control in order to remain competitive on the Internet access market (*i.e.* maintaining low pricing/level of margin). They also want to control the global QoS offered to certain services and to maintain a certain level of differentiation.

Actors include ISP, transit network operators, content providers and CDN. In short, content providers want to be notified of each action of their clients. Therefore traffic interception by un-authorized third-parties is not acceptable. On their sides, CDN providers have deployed a content delivery infrastructure (servers, possibly backbone networks) and have agreements with content providers. Finally, ISPs want to maintain cost structures under control and to control the global QoS offered to certain services.

The explosion of multimedia traffic recently changed the market equilibrium. Since CDN providers have deployed most of their servers out of ISP networks, bottlenecks start appearing in the peering links between ISP and CDN networks, which has led to a series of clashes between well-established actors [ora12, Gol10]. To overcome this problem, CDN providers would like to deploy more repositories directly in ISP networks, while ISP promote the raise of so-called *telco-CDN*, which consists of multiple ISP-controlled repositories with guaranteed links to the customers [Ray09, LS13]. However, telco-CDNs focus on a small area although content providers generally need global delivery. This is an opportunity for traditional CDN providers, which can coordinate multiple localized telco-CDNs and interface these latter to the content providers. Moreover, CDN can also profit in sharing their experience of content delivery with ISPs through licenses or co-management agreement [lcd12]. This analysis leads to the proliferation of repositories controlled by third-party actors within the ISP networks. In a word, I believe in multiple independent repositories deployed within ISP networks. The management of these repositories can be done in multiple ways, including (*i*) by the ISP itself, (*ii*) jointly by the ISP and a CDN provider through a licensing agreement, and (*iii*) by a renting CDN provider through a collocation agreement. I also believe that network operators will keep control on traffic management and will look for solutions to smoothly integrate repositories, including information-centric protocols.

5.2.2 Naive Integration of Repositories in CCN

So far, repositories have been absent of CCN protocol, as well as of other information-centric network proposals. A very recent version of CCNx software features a so-called *repository* component, which is expected to let applications save some data in the local hard-disk of routers [WWWe]. This first step toward the integration of CDN repositories in CCN is however not documented well from a network standpoint [WWr]. Nothing is said about the integration of such router component in the protocols.

In CCN the FIB is filled with the broadcast content advertisements, *e.g.* by using the OSPFN [WHY$^+$12] protocol. Thus CCN nodes are aware of available contents and can implement routing protocols. FIB are key elements of information-centric routing, and how to fill FIB is critical for our problem. There are four naive ways to integrate repositories in CCN through the FIB, but none of them is good. Here are why.

- To better integrate CDN repositories into CCN, one solution is to make repositories advertise what they store with the full name. This solution however does not scale: To advertise every content with its full name would induce network overhead and huge FIBs. Such solution can work with a small number of contents but it cannot be widely implemented.

- Another solution is to make CDN repositories advertise only the domain names of their clients, so that *all* the requests for the same domain are forwarded toward CDN repositories. Such solution respects constraints on FIB memory size. However, the CDN caches hold only a subset of the catalog offered by the service providers, then if an *Interest* message is sent towards a repository that does not have the content, no intermediate entity is aware of this miss (no negative response in CCN). Worst, all future retransmitted *Interest* packet also fall in the trap: they follow the same path to the same repository, which cannot reply.

- Another solution is to lever native multicast features of CCN. A FIB supports multicast by default by associating several outgoing information to an FIB entry. For the domains that are included in the CDN repository, it is possible to still maintain the information about the original server so that the problem encountered above can be solved by simply forwarding *Interest* packets twice: one to CDN repository and one to original server. This solution is however not efficient. The original server receives all requests although most of them can be handled by repositories. Such solution is against the original intention of CDN services and double Interest-related traffic.

- The last idea is to add all CCN node functionalities and components into the CDN repository and transform it into a transit CCN node with an extra-large Content Store (e.g., the *CCNx Repository* element). If such CDN/CCN hybrid node experiences a miss, it would simply forward the *Interest* packet as in a regular CCN nodes, based on its PIT table and FIB. I discarded this idea because (1) it would impact the networking topology (lot of traffic would have to pass through this CDN/CCN node, with the risk of creating bottlenecks) (2) the repository should perform not only a fast content management, analysis and retrieving process, but also a fast *Interest* packet management and redirection performance, which is not viable as proved by [PV11], and (3) since such CDN/CCN is on most forwarding path, a failure of the repository function, for any reason, would result in a loss of all *Interest* packets in the area.

5.3 Introducing cRouters

I now introduce the cRouter. A cRouter is *associated* with at least one CDN repository. A cRouter has all functionalities of a regular CCN node and is also connected to other CCN nodes. It has two new components (Figure 5.1):

- *Repository Forwarding Table* (RFT), which stores the domain names of the content providers that have agreements with the associated repositories

- *Pending Repository Interest Table* (PRIT), which stores the *Interest* messages that were previously forwarded to one of the associated repositories

5.3.1 The Repository Forwarding Table

In CCN, a FIB table contains the *Interest* packet forwarding information. This table is filled by the content publications that are advertised by servers. I define a RFT

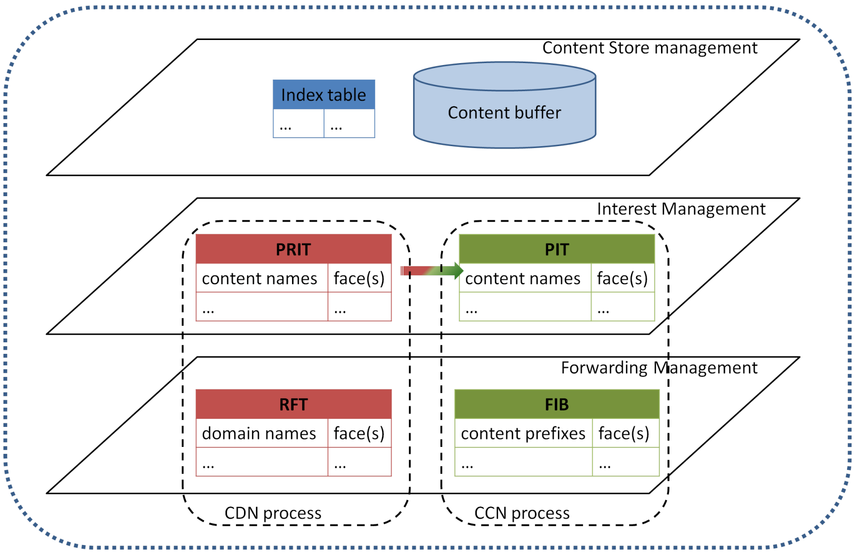


Figure 5.1: The CCN node architecture with the repository elements

as a FIB-like table, which is used to store only the content domain names that are served by the repositories which are associated with the node. As in a FIB, a RFT contains the outgoing face identifiers through which the CCN node can access the right repository. I give an illustration of a RFT in a cRouter in Table 5.1. This table shows that this cRouter is associated with at least three repositories. Two of them are in charge of serving videos from *Youtube* and *Dailymotion* and the remaining one has agreements with *Google*.

Repository Forwarding Table	
Domain names	**Faces**
YouTube, DailyMotion	101, 102
Google	103
. . .	. . .

Table 5.1: Repository Forwarding Table

To fill in the RFT, we see two options: (*i*) the ISP operator that is responsible of the cRouter fills its RFT. This option is for ISPs that know well their own CCN topologies and the repositories that are deployed in their networks, no matter these repositories are managed by themselves or by third-party providers. Although the set of contents that are actually stored in a repository can be dynamic, the domain names of these contents are generally defined when the repository is installed, and are relatively stable. Furthermore, it is expected that the ISP knows well the configuration of cRouters and their associated repositories, hence RFT tables can be manually filled by ISP engineers themselves. (*ii*) the RFT is filled in the same way as FIB tables are filled, *i.e.* through the implementation of a broadcasting protocols like OSPFN. But only the domain-name level name prefixes are published in the OSLA packets [WHY+12]. I typically propose to add a flag in OSPFN packets to differentiate the publications to FIB from those to RFT. When a cRouter gets a RFT-related publication, it creates or updates a RFT entry with the domain names and the face identifier from which this cRouter received the advertisement. A regular CCN node receiving such RFT-related publication can ignore it. In the following, I detail RFT processing together with PRIT processing, as illustrated in Figure 5.2.

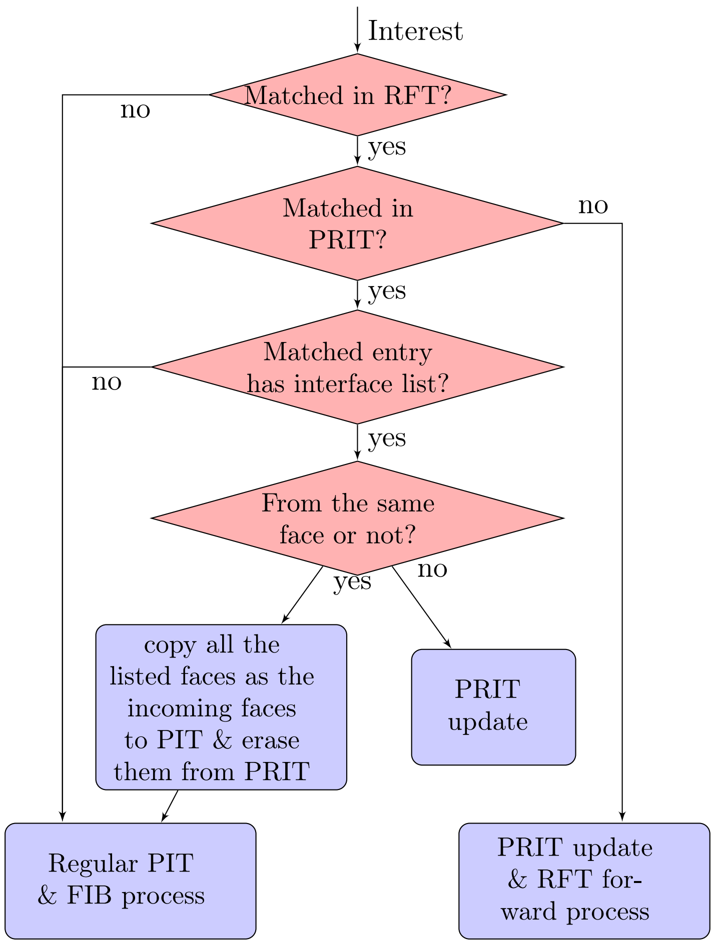

Interest
Matched in RFT?
no
yes
Matched in PRIT?
no
yes
Matched entry has interface list?
no
yes
From the same face or not?
yes
no
copy all the listed faces as the incoming faces to PIT & erase them from PRIT
PRIT update
Regular PIT & FIB process
PRIT update & RFT forward process

Figure 5.2: The *Interest* message process

5.3.2 The Pending Repository Interest Table

In this work, a repository is a storage server, and it does not implement any redirecting function, which is not supported in CCN. Thus, I have to implement a special mechanism to handle the case of miss at the repository. I propose the PRIT, which is somehow similar to the regular PIT table, but only applies to the *Interest* messages that have been forwarded to repositories in the past. The role of the PRIT is to allow the detection of the incoming *Interest* messages that have probably generated a miss at the repository. I give an example of PRIT in Table 5.2.

I detail the processing of incoming *Interest* packet in Figure 5.2. When an *Interest*

Pending Repository Interest Table	
***Interest* ContentNames**	**Incoming faces**
ccnx:/youtube.com/music/top10/ GamnanStyle.flv/chunk0	01, 04
ccnx:/google.com/web/gamnan %+style%+live/	

Table 5.2: Pending Repository Interest Table

packet arrives at a cRouter, the RFT lookup step is added just before the regular PIT-FIB process. After the Content Store check (if CS does not have the content), the cRouter looks for the segment of the *Interest* domain name up in the RFT. If the lookup result is negative, then the cRouter enters to the regular PIT-FIB process. On the contrary, if the RFT gives a positive check, the cRouter should firstly try to get the *Data* object from the right CDN repository and the cRouter enters in the process related to the PRIT. There are four cases:

- **No matching entry in the PRIT**. It is the first time the cRouter receives an *Interest* message for this ContentName. The *Interest* message is forwarded to the repositories according to RFT information. Meanwhile, a PRIT entry is created with the ContentName and the incoming face identifier. For example in Table 5.2, when the *Interest* message for *Gamnan Style* video provided by *Youtube* first arrived from face 01 at the cRouter, a new entry has been created for this entry (there is no related entry created in the PIT table).
- **A matching entry in the PRIT but the incoming face is not within the associated face list**. The cRouter has already forwarded a similar *Interest* packet to a repository and is still waiting for the *Data* packet. The PRIT only updates the entry with the new incoming face. For example in Table 5.2, when the same *Interest* request for *YouTube*'s *Gamnan Style* video arrives from face *04*, the matched PRIT entry is updated with face *04*, but this *Interest* packet is not sent out.
- **The incoming face is in the face list which is associated with a matching entry**. In CCN, the user sends again the same *Interest* message when it does not get the content after a time-out. Here the cRouter receives again the same *Interest* message from the client side, and finds a matching PRIT entry for both this ContentName and this incoming face. The cRouter thus realizes that a CDN cache miss occurred (the cRouter has sent a same *Interest* message to a repository earlier but the CDN node did not reply) and this *Interest* packet should be sent to the original server according to the regular FIB. The entry in the PRIT is kept *without any face* (see for example the second entry in Table 5.2), so that future *Interest* packet will not be forwarded to repositories. The *Interest* packet is then forwarded to the original server.
- **A matching entry without any associated face**. The *Interest* message for this ContentName has already been forwarded to original server because

the repository was unresponsive. For example if the cRouter receives again an *Interest* request for *Gamnan Style Live* in *Google*, this *Interest* packet should be directly processed with the regular PIT and FIB processes.

Each PRIT entry also implements a timer (as in PIT). The role of the timer is to clean the time-out entries that have not found the content in the repository but no subsequent similar *Interest* message has been received. The PIT timer is short, which is in the order of the *Data* packet round-trip-time. The PRIT timer should be set longer (for example in the order of minutes or hours), because CDN repository management policies are dynamic.

5.3.3 Data Packet Processing

The processing of *Data* packets at a cRouter is less complex. It is presented in Figure 5.3. When a *Data* packet is returned back to a cRouter, whether this *Data* packet comes from the repository or from a regular server should be firstly distinguished. If the *Data* packet comes from a regular server, the regular PIT lookup process is performed. The related PRIT entry (if it has any) is not deleted so that the following *Interest* messages will not be sent to the repository again. If the *Data* object comes from a repository, only a PRIT lookup is performed. Thus, the *Data* message is sent to all faces that are associated with the ContentName in the PRIT, and then the entry is deleted. Please note that it may occur that the entry does not contain any associated faces, for example when the repository has been very long to emit a *Data* response and in the meantime the cRouter received a re-sent *Interest* packet. In that case, the PRIT entry should be deleted.

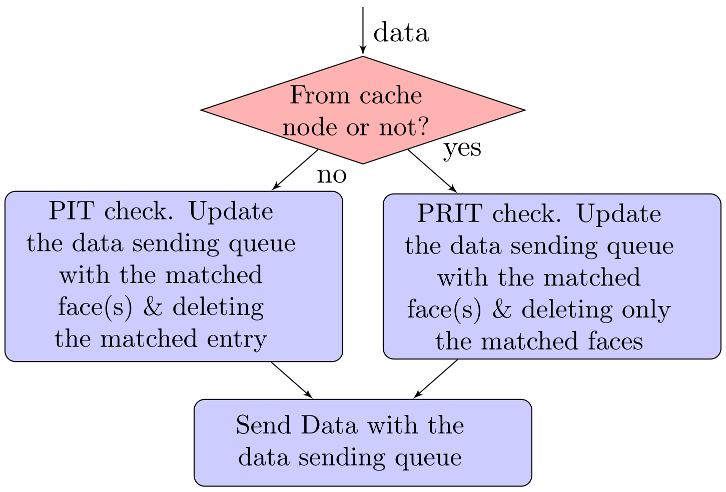

data
From cache node or not?
no
yes
PIT check. Update the data sending queue with the matched face(s) & deleting the matched entry
PRIT check. Update the data sending queue with the matched face(s) & deleting only the matched faces
Send Data with the data sending queue

Figure 5.3: The *Data* message process

5.3.4 Packet Processing in the cRouter

In this section I present the *Interest* and *Data* packet processing algorithm, described with the different use cases.

The *Interest* packet process with the different use cases is as follows:

- Use Case 1: **The *Interest* domain name is not present in the RFT (CDN does not cache content for this domain)**. The cRouter simply processes the *Interest* packet with regular PIT and FIB tables in order to forward the *Interest* to the original server (as shown in Figure 5.4).

- Use case 2: **The requested content is cached in the CDN repository**. It is the first time the cRouter receives an *Interest* message for the ContentNamen ad there is no PRIT entry. The *Interest* message is forwarded to the repository according to RFT. Meanwhile, a PRIT entry is created with the ContentName and the incoming face identifier (as shown in Figure 5.5).

- Use case 3: **A matching entry in the PRIT but the incoming face is not within the associated face list**. The cRouter has already forwarded a similar *Interest* request to a repository and is still waiting for the *Data* object. The PRIT only updates the entry with the new incoming face. (as shown in Figure 5.6).

- Use case 4: **The incoming face is in the list of faces associated with a matching entry**. The cRouter receives again the same *Interest* packet from the client side, it finds a matching PRIT entry for both this ContentName and this incoming face. The cRouter thus realizes that the CDN repository did not reply for this content and a miss occurred. The *Interest* packet is then forwarded to the original server according to the regular FIB. Moreover, the list of faces that was associated with the entry in the PRIT is now stored in the regular PIT, thus all previous *Interest* messages for the requested content will be fulfilled when *Data* message will get back from the original server. The entry in the PRIT is kept in the table but *without any face* (we removed all associated faces). It thus indicated that a miss occurred and that future *Interest* request should not be forwarded to repositories. Later if an *Interest* packet for this ContentName comes again, it will find a matching entry without any associated face and the *Interest* message will be directly processed with the regular PIT and FIB processes (as shown in Figure 5.7).

- Use case 5: **The CDN repository lately replies with a content**. This case means the CDN repository has the requested content but for whichever reason it replies late after the cRouter considered the case as a CDN miss. The first steps are the same as in use case 4, and when the delayed response from the CDN repository arrives at the cRouter, it will find a matched PRIT entry with an empty face list. In this case it will simply delete the related PRIT entry (as shown in Figure 5.8) to indicate it is not a miss and subsequent *Interest* messages can be forwarded to the CDN repository.

The processing of *Data* packets at a cRouter is less complex. When a *Data* packet is returned back to a cRouter, we should first distinguish whether this *Data* message comes from the repository or from a original server. If the *Data* message comes from a original server, the regular PIT lookup process is performed. The related PRIT entry

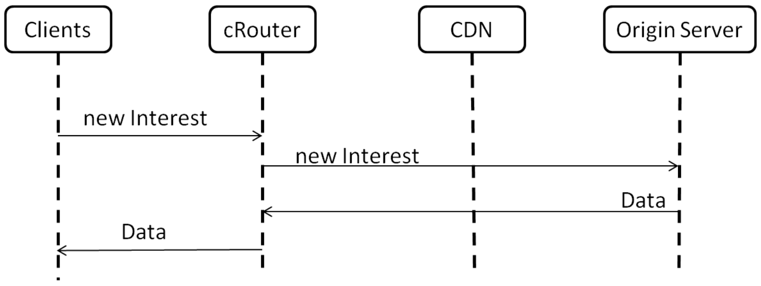


Figure 5.4: Use case 1: The *Interest* domain name not present in RFT (this domain name is not managed by the CDN repositories.

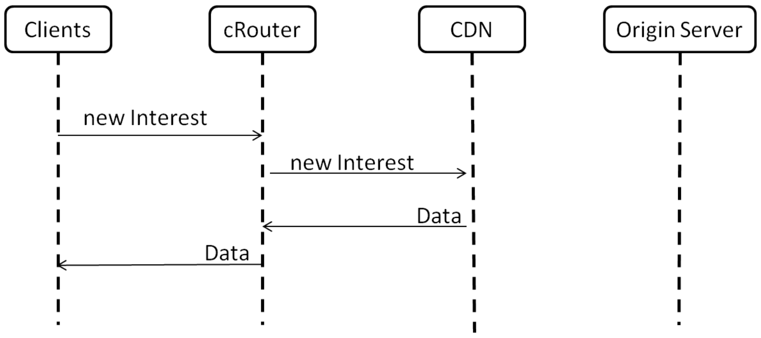


Figure 5.5: Use case 2: *Interest* domain name present in RFT, but content name not in PRIT. CDN repository has this content.

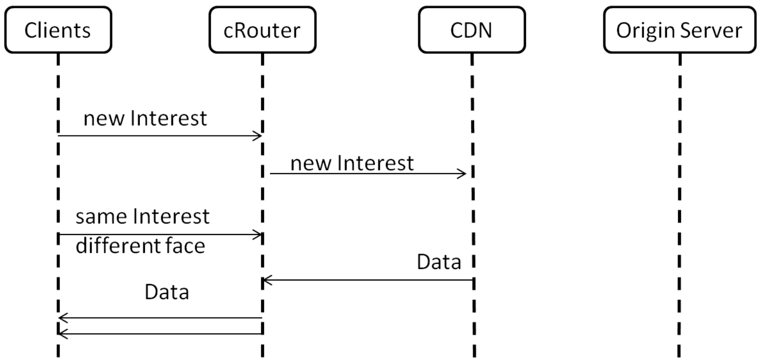


Figure 5.6: Use case 3: Same *Interest* from different clients arrive, domain name present in RFT, CDN repository has this content.

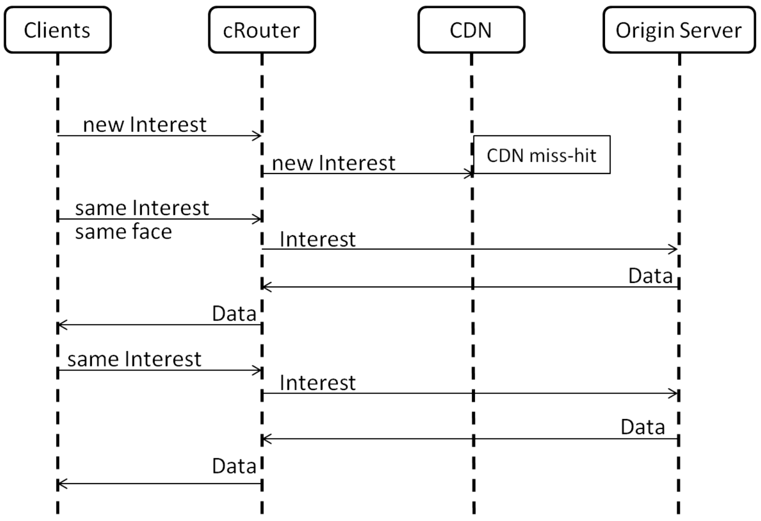


Figure 5.7: Use case 4: CDN repository miss (domain name managed by the CDN repository but content not cached).

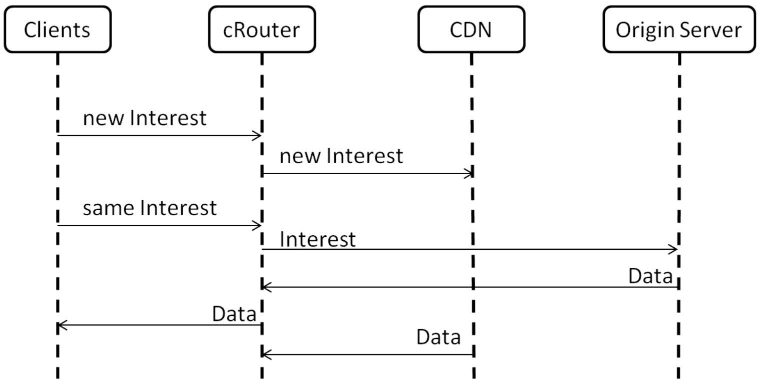


Figure 5.8: Use case 5: CDN repository replies lately

(if it has any) is not deleted so that the following *Interest* messages will not be sent to the repository again. If the *Data* message comes from a repository, only a PRIT lookup is performed. Thus, the *Data* packet is sent to all faces that are associated with the ContentName in the PRIT, and then the entry is deleted.

5.4 Example of cRouters Deployment

I present in this section an example of plugging CDN repositories in cRouters. Please refer to Figure 5.9 for an illustration.

Suppose an operator with a three-level hierarchical CCN networking topology. In Figure 5.9, the links between CCN nodes are continuous. This ISP would like to host some CDN repositories in its network. Two CDN repositories are deployed, say *CDN repo A* and *CDN repo B*, which store contents from *Youtube* and *Dailymotion*, respectively. The ISP has observed that the best places to "plug" these repositories

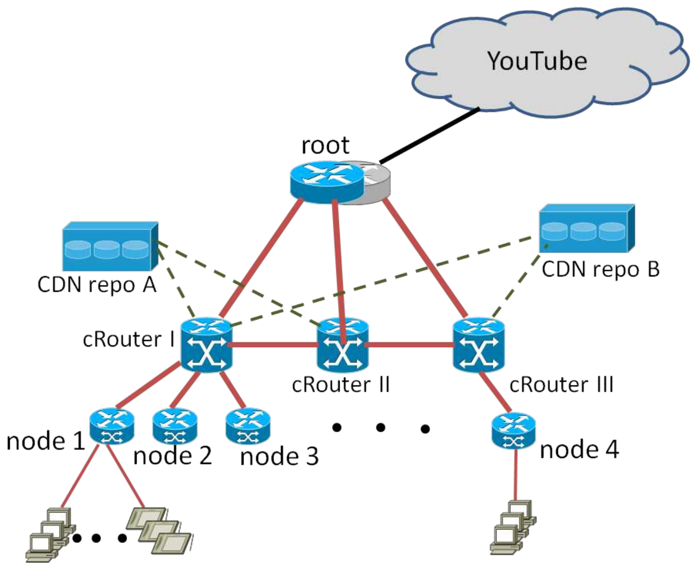


Figure 5.9: Example of cRouters in a hierarchical network with CDN repositories

are at the regional-level routers. This decision considers the volume of traffic and the characteristics of the repositories (storage size and bandwidth capacity). The repositories are connected with the regional-level routers by the dotted lines. The *Youtube* server is outside the ISP domain and it is connected with the ISP root router. I deploy the cRouter solution on the regional-level nodes because these are the only CCN nodes that are connected to repositories.

FIB of cRouter I	
ContentName prefixes	**Faces**
youtube.com	01 (root)
...	...

Table 5.3: The FIB table of cRouter I

RFT of cRouter I	
Domain names	**Faces**
youtube.com	11 (repository A)
dailymotion.com	12 (repository B)

Table 5.4: The RFT table of cRouter I

Let us take *cRouter I* as an example. Due to the regular advertisement publication from *Youtube* server, the FIB of *cRouter I* looks like Table 5.3. The *cRouter I* is also manually configured to link with the CDN repositories *A* and *B*. Its RFT is represented in Table 5.4. The nodes *1*, *2*, *3* and *4* are regular CCN nodes, they are not cRouters.

Suppose now *cRouter I* receives from node *1* an *Interest* request for a popular video in *YouTube*. *cRouter I* firstly checks its RFT and finds a matched entry with

CDN repo A because this repository is in charge of all *Interest* packets for *YouTube* (see Table 5.4). Meanwhile no entry matches in the PRIT. Therefore, this *Interest* packet is forwarded to cache *A* and meanwhile an entry is created at the PRIT at *cRouter I*. *CDN repo A* stores this video and returns the file to *cRouter I*, and subsequently to the end-user following the reverse path. Later *cRouter I* receives from node *2* an *Interest* packet for a non-popular *YouTube* video. Similarly, a related entry is created in the PRIT and the *Interest* message is forwarded to the *CDN repo A*. Unfortunately *CDN repo A* does not store this video. Thus *cRouter I* does not get any response. Then the end-users resends its *Interest* request. Now the ContentName matches an entry in PRIT of *cRouter I*, with the same incoming face. The *cRouter I* knows it has already required this content to *CDN repo A*. Then it keeps the PRIT entry with an empty list of face, and puts this list in a new PIT entry. The *Interest* packet is forwarded to the *root* node according to FIB. When the video *Data* object is back from the regular *Youtube* server and arrives at *cRouter I*, *cRouter I* knows the *Data* packet is from the original server not from the CDN repository. Then only the PIT lookup is triggered and the PRIT entry still shows that this file is not stored in the repository.

5.5 Evaluations

To the best of my knowledge, no previous works has addressed the interworking of the CDN into the CCN. Hence I do not compare the cRouter solution to others, rather provide an evaluation of some key characteristics of a cRouter.

5.5.1 Memory requirement analysis

A cRouter is a CCN node that contains two new tables. First of all it is interesting of measuring the extra-cost of these memory spaces and of analysing the feasibility in terms of the memory requirement. I focus on the PRIT in this discussion, because the RFT stores only a few domain names. According to studies [GFMM12], a large portion of the traffic volume is related to a dozen of domain names, so the size of RFT is just in the order of hundreds Bytes, which can be implemented on any fast memory chip.

Let us start with a description of a typical cRouter. It is connected with the CDN repositories through the line cards of 20Mbps. The *Data* object *Round-Trip Time* for in-domain networking and for out-domain are 20ms and 150ms respectively [SKBT12]. I consider that 65% of the incoming traffics can be forwarded to a repository because the requested files are in the domain name handled by the CDN [GFMM12]. I assume that each *Interest* packet has 40-Byte length and the average length of ContentNames in *Interest* messages are 30-Byte. I add 2-Bytes for the face identifiers in PRIT entry [PV11].

I consider today's memory chip technologies (see Table 5.5 for a summary from [PV11]) and two possible implementations of the PRIT. The most basic implementation is based on a hash table, while a more sophisticated implementation is based on Bloom filters [YMT^{+}12], as I presented in Chapter 3. I applied a counting Bloom filter which has five independent hash functions and the designed false positive rate is 0.1%.

Technology	time (ns)	Cost ($/MB)	Max. size
SRAM	0.45	27	≈ 210 MB
RLDRAM	15	0.27	≈ 2 GB
High-speed SSD	1,000	0.03	≈ 10 TB

Table 5.5: Current memory technology [PV11]

I represent in Figure 5.10 the memory requirement of PRIT in function of the CDN hit ratio. The hit ratio plays an important role. The higher the hit ratio is, the lesser memory space is required. The reason is as follows. The in-domain latency between the CDN repositories and the ISP routers is smaller than the out-domain latency between an ISP peering routers and the original content servers (for example *Youtube* is in an isolated Autonomous System). Thus according to *Little's Law*, the more requests are treated by repositories, the shorter *Interest* packets stay in the PRIT, consequently the table size is smaller. We also can see that both the hash-table based and the Bloom filter based PRIT solution can be built with the fast SRAM memory chip.

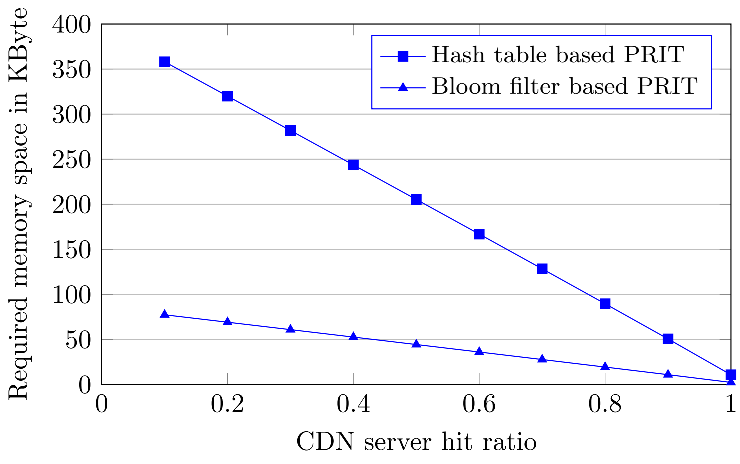


Figure 5.10: The memory space requirement vs. the cache hit ratio

In Figure 5.11, I present the total memory requirements after the implementations of PRIT plus the current PIT in comparison to the size of the original single PIT without the PRIT. I compute the ratio of the total size requirement to the original PIT size. When the curve is above 1, the total size of PRIT plus current PIT is larger than the original single PIT. The higher the hit ratio is, the more memory savings we get, since the *Interest* messages with a matching *Data* object from CDN repositories are no longer pending in the PIT table. However, the miss-hit *Interest* should wait for the second emission to reach the original server, which costs longer time. Indeed if the hit-ratio is low, there is more *Interest* messages pending in the PRIT and the required memory space is larger.

Then the impact of traffic distribution on the memory requirement is analysed here. I suppose the incoming traffic follows the Zipf distribution. The Figure 5.12 shows the PRIT memory size in function of different Zipf distribution parameter (α).

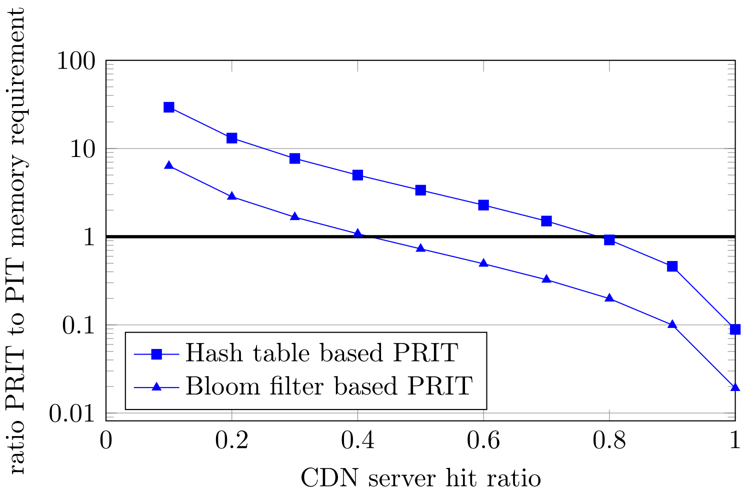


Figure 5.11: ratio of PRIT to PIT memory requirement vs. the cache hit ratio

In this evaluation I assume the incoming traffic rate is 10Mbps, 20Mbps and 50Mbps. As we can expect, the memory requirement is sensitive of the incoming traffic rate (the memory size of a higher *Interest* packet rate is larger than the one of lower traffic rate), and also the content popularity (for example for each incoming rate, the memory requirement of $\alpha = 1$ is 50% of the memory at $\alpha = 0.2$).

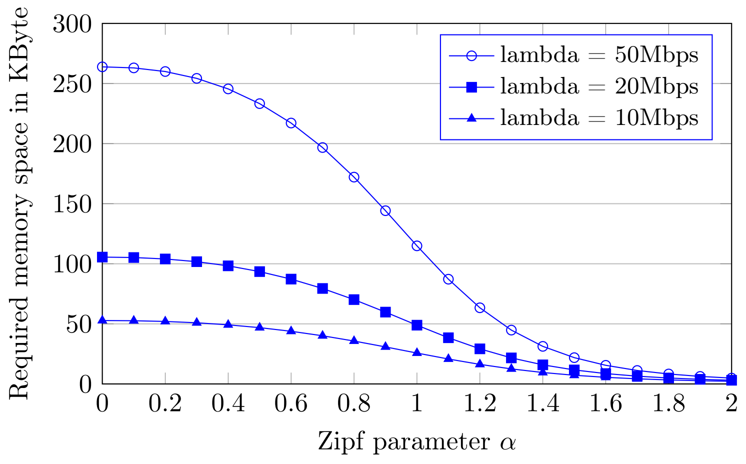


Figure 5.12: PRIT size vs. zipf distribution

5.5.2 Redirection rate vs. content popularity

The redirection rate is the ratio of the number of the *Interest* messages which are redirected to the original server by the PRIT in case of a miss-hit at the CDN repositories or because of the lack of PRIT space, over the number of the total *Interest* messages. The redirection rate should be limited as low as possible because high redirection rate means that more *Interest* packets are resent to the original sources so that more networking traffic are generated and these *Interest* messages will suffer a longer *Data* response time. From Figure 5.13 we can see that implementing a larger size PRIT can significantly reduce the *Interest* packet redirection rate. I notice also that the content popularity gives a evident impact on the redirection rate. The uniform distribution raise a higher redirection rate. For example even with a 96KByes, the redirection rate is 20% for $\alpha = 0.2$. But as long as the content popularity becomes less uniform, the redirection rate is significantly reduced. The 96KBytes PRIT suffers 6% redirection ratw when $\alpha = 0.8$, which refers the generic web traffic [FRRS12].

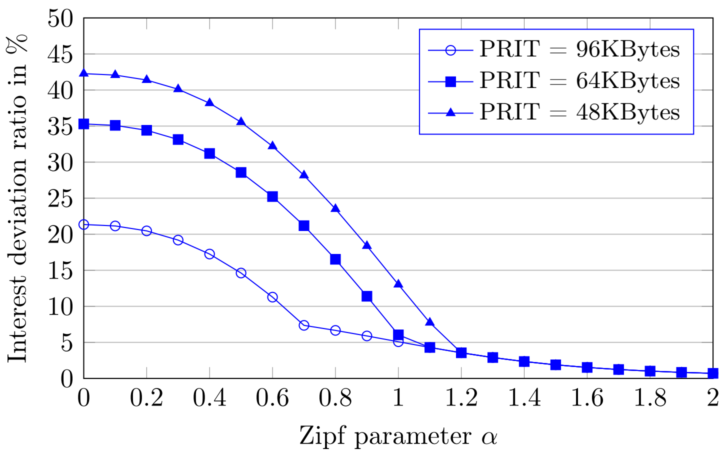


Figure 5.13: Redirection rate vs. zipf distribution

5.5.3 Data Response time analysis

One of the key value of the CDN deployment is to offer a better quality of experience, for example a faster content delivery time. We now analyze *Data* response time issue. Most settings are similar as in Section 5.5.1.

To reduce the impact of a miss in the repository, one solution is to set a shorter *Interest* message retransmission interval. However, such setting would lead to a significant growth of the number of networking message. In Figure 5.14 I set the hit ratio at the repository at 0.7. We can see that a shorter *Interest* message retransmission interval can reduce the average *Data* object response time but it introduces more extra interworking messages. Suppose we do not want have more than 0.5 extra messages, then the *Interest* message retransmission interval should be no longer than 200ms. With such reasonable setting of 200ms *interest* message retransmission interval, the average *Data* response time is near 120ms.

5.5.4 Location of deployment analysis

I consider now the topology of Figure 5.9. For an ISP, CDN repositories can be located either at the regional routers level (for example *cRouter I*) or at the edge routers level (for example *node 1*), which are closer to the end-users. Since regional routers experience a higher *Interest* packet arrival rate, the configuration of cRouter

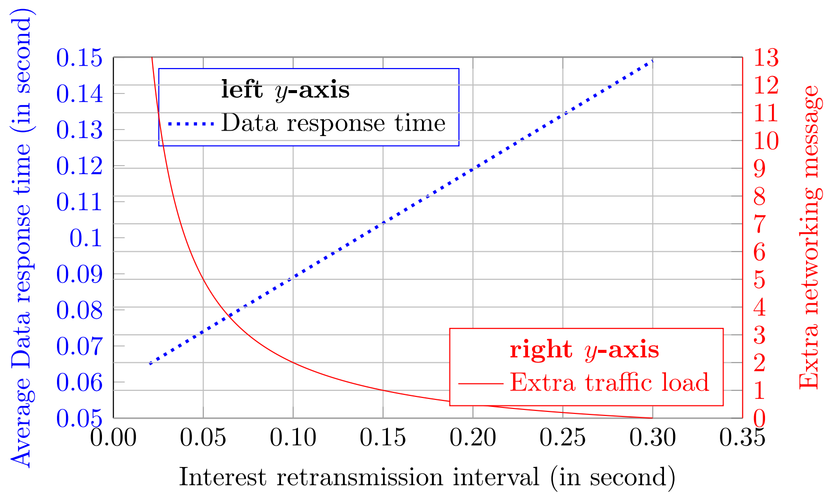


Figure 5.14: *Data* packet response time and extra traffic load vs. *Interest* message retransmission interval

should include fast but expensive memory chips for PRIT. The advantage is that the efficiency of repositories is better. On the contrary, RLDRAM can be enough if cRouters are deployed in edge routers level but more repositories are required.

In the following evaluation, I set the *Interest* packet arrival rate of *edge node* (for example *node 1*) in the range of $1 - 10$Gbps. A repository is equipped with a 1TB high-speed SSD disk as the content container. The other parameters are the same as in the previous sections. The configurations are as follows: in the *edge cRouter scenario*, cRouters are deployed at edge nodes, while in the *regional cRouter scenario*, cRouters are in regional nodes.

I am also interested in evaluating the cost of the PRIT memory to handle such traffic. as illustrated in Figure 5.15, until an *Interest* packet arrival rate 7Gbps, the regional cRouter scenario is cheaper because, in both scenarios, cRouters are equipped with RLDRAM-based PRIT but the overall memory size of cRouters at the regional level is smaller than at the edge level. However, when the *Interest* packet arrival rate exceeds 7Gbps, the regional cRouter scenario becomes much more costly because the regional node collects all the traffic from the edge levels and the traffic rate becomes so high that SRAM technologies should be implemented on the routers.

5.5.5 Discussion

The results of these evaluations show that even if this solution requires two additional tables, the whole memory requirement of the total forwarding tables does not increase much. The performance of the cRouter depends on the content behaviours, for example the content popularity. In most networking environment, the content popularity parameter α of the Zipf distribution is greater than 0.8, and in this condition the memory requirement is at the same level as the classic CCN node design, the packet redirection rate is limited at a low level (*e.g.* 2% for 96KBytes PRIT). A shorter average Data response time requires a shorter Interest packet retransmission interval

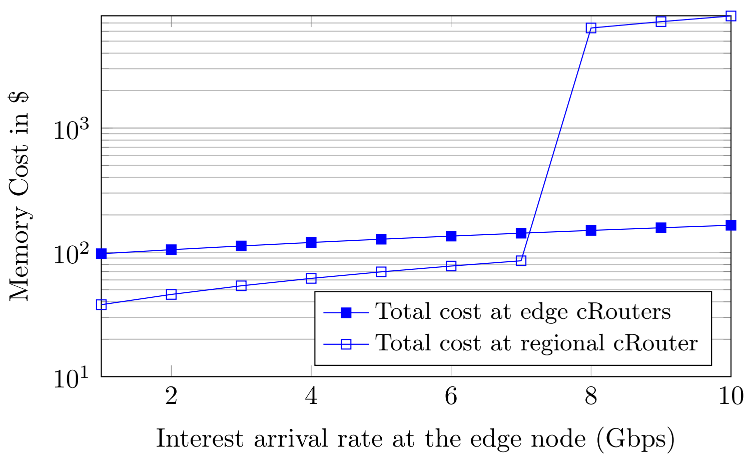


Figure 5.15: The cost of different deployment location

in order to quickly recover the CDN miss-hit. However that will lead to extra networking traffic. So during the realistic implementations, a trade-off should be made between the networking traffic and the Data response time performance. For example if we want to limit the average Data response time for 70ms, we have to suffer about two times extra networking traffic.

5.6 Conclusion

The advantages of CDN service are well known in IP. It brings a better end-user quality of experience and reduces the traffic cost of both content providers and ISP operators. Meanwhile the CCN networking paradigm emerged, with some use-cases to show the feasibility of the approach, but did not take into consideration CDN networks so far. But I believe that the deployment of CCN should not ignore these key advantages of CDN. In this contribution, I analyzed that the current CDN design can not support and interconnect with CDN networks and an evolution of the CCN node interconnected to the CDN repository, is necessary. That is why I have proposed the cRouter, a "pluggable" CCN node with two new tables, namely the RFT (Repository Forwarding Table) and the PRIT (the Pending Repository Interest Table). The RFT can differentiate the forwarding information between original servers and CDN repositories. The PRIT can tolerate the *Interest* packets redirection problem in case of a repository miss. The evaluations that I have performed show that including the two new tables has no additional cost for building a CCN node and the flow of messages is not significantly affected. I have also exemplified how the cRouter can be deployed in a real hierarchical network of an operator. The cRouter solution is fully compliant with current CCN design (*i.e.* the cRouter can be connected to current CCN node) and allows to keep the relationships and positions of key players in the delivery chain: The CDN providers still keep the control of the storage functions, while the network operators still focus on the transport part of the content.

Chapter 6

Conclusion

The IP network architecture has been designed in 60s-70s for a communication-based usage. But this end-to-end model can not perfectly match the Internet usage evolution which is more oriented to a content-based model. The Content-Centric Network (CCN) has been proposed in 2009 by Van Jacobson and his team of PARC, with the purpose to replace the IP based Internet architecture by a content-based one, in order to improve the delivery quality. The CCN integrates many services directly into the networking design such as multicast, security, in-networking caching, *etc.*, and brings many advantages for the network of the future. But the current hardware technologies, especially the memory chip technologies, are not yet ready for supporting the switch from a IP address based network to a content name based one. In this thesis I investigated the realistic design and implementation issues of the CCN design. The thesis is composed by three main contributions, summarized in the following:

- **A distributed PIT based on Bloom filters:** This contribution investigated the CCN PIT element. I focused on this component because PIT is a brand-new element in CCN that does not exist in IP architecture. I was then interested in analysing the behaviour of this element. During the forwarding operation, PIT is involved in both incoming *Interest* and *Data* packet processing. Thus PIT should perform fast enough to follow the incoming packet rate. Meanwhile, *Interest* and *Data* perform exact matching in PIT. That means PIT should memorize the entire ContentNames of the incoming *Interests*. This behaviour leads to a large memory space requirement. Unfortunately today's memory chips can not meet these two needs. The fast memory, for example SRAM, does not have a large volume. Or the huge memory chip which has a large space capacity, for example RLDRAM, can not offer a fast access time. In order to overcome the hardware bottleneck and keep the original PIT functionalities, I proposed a distributed PIT structure which is based on the Bloom filter solution (named *DiPIT* system). In my proposal the Bloom filter solution is applied with the purpose of reducing the memory requirement. Furthermore, instead of having one single centralized PIT in each CCN node, I distributed the PIT to each CCN face. Each distributed small PIT (named *PITi*) was in charge of the packets only for the face which it is associated to. The distributed system has two advantages. First of all, the size of each single PITi is smaller. It thus can be easily implemented into the fast SRAM. Secondly, the distributed design can resolve the face information retrieval issue of the Bloom filters. The evaluation results showed that the Bloom filter based PIT solution can significantly reduce the memory requirement. The implementation of CCN PIT then becomes possible facing today's memory limitations. Even if, in the future, the memory technologies evolve, which means the fast memory can support relatively larger volume, this proposal is also interesting because it costs always less memory than the traditional method (*e.g.* hash table based). This solution is proposed for the PIT element, however the same idea can also be applied to others, for example the FIB table.

- **A dynamic content-aware forwarding system:** The Content-Centric Networking requires the content based routing, relying on the content name based Forwarding Information Base (FIB). However in current CCN design, the CCN FIB is filled by the flooded content advertisements, which is similar to the IP address announcement. This design has some limits. First, since each content should be announced through the network and since the number of contents is much bigger than the number of current IP addresses, the content advertisements will introduce lots of networking traffic. Second, in this traditional method, each node which receives the content advertisement has the same view of all the available contents in the networking. That will cause a huge FIB table size. Thirdly this design does not take into consideration the content behaviours, such as the local content popularity. So for a network which is based on the content itself, it is neither efficient nor optimized. At last since in this method the FIB only passively receives the content advertisements from the content provider and in CCN the on-path caches do not announce the contents that they have, so the Content Stores or other content cache capacities at the end users' equipment are actuality not included in the conventional forwarding design. Focusing on these four points, in this contribution I proposed a dynamic content-aware CCN forwarding design. This design contains three contributions: a content publication protocol, a content aware FIB filling algorithm and a dynamic downstream forwarding component. This solution is really content based, considering the content and end-users' behaviours. The content advertisements are propagated only through certain paths and towards certain nodes, instead of flooding the entire network. The FIB is filled based on the incoming contents, and it memorizes only the locally popular content forwarding information. The dynamic downstream forwarding element is built based on the PIT. It considers the end-equipments also as the content providers for *Interest* forwarding. It is aware of the caches of the downstream nodes and can forward the matching content requests to those nodes in order to discover the best content containers. The mentioned downstream nodes can be the Content Store of the on-path nodes, or the caches at the end-users' equipment in a P2P system, at the users' set-top boxes, or the game terminals, *etc.*

- **The interconnection of CDN services with CCN network:** The Content Delivery Network (CDN) has already proved its success in IP network. CDN has brought many advantages in both technical and business aspects. The

CCN can provide the in-networking caching but since the main usage of this kind of caches are fundamentally different from the CDN repository servers, we cannot simply and totally use CCN to replace CDN. And of course we still want to keep the advantages of CDN in any new generation internet proposals, including CCN. However the features from the original CCN design does not support the interconnection between CCN network and CDN services. For example the no-DNS reliable feature does not support the redirection from the original server to the CDN repositories. The no-address/location behaviour can not support the packet redirection neither, especially in case of a CDN content miss. In this contribution a CDN forwarding mechanism and an *Interest*/*Data* process protocol are proposed, in order to make the CCN structure able to support the current CDN service. In this solution, two new tables are added in a CCN node. One table, named RFT (Repository Forwarding Table), is in charge of deciding which *Interest* messages should be forwarded to the CDN servers. It contains the forwarding information of the domain names which are managed by the CDN repositories. The second table, named PRIT (Pending Repository Interest Table), has two functionalities. The first one is for *Data* packet forwarding. Similar as PIT, when the CCN node forwards an *Interest* message towards the CDN repository, the *Interest* is appended in the PRIT table for the *Data* packet forwarding later. The second role of PRIT is for the awareness of content miss at the CDN servers. If the CDN repository does not have the required content, according to the CCN design, the CCN node will receive a same repeated *Interest* message from the client side, and then find an existing entry at the PRIT table. Thus it can notice that this *Interest* message has already been forwarded to the CDN repository earlier and it can not be satisfied by the CDN repository. The CCN node then should resend the *Interest* message to the original servers.

The three contributions focus individually on the three main CCN node components: PIT, FIB and CS. Each contribution is trying to improve the performance of the different elements, and also to make the realistic implementation of each element possible facing today's hardware limits. These three contributions are also coherent. Indeed, we can combine all the solutions together and build a whole optimal CCN node that has a better performance and can meet the current memory technology limits. But the solution proposed in one contribution can also be used in the others. For example the distributed Bloom filter structure which is originally proposed in DiPIT can also be applied in the dynamic FIB system.

In one word, in this thesis, I focused on the different CCN node elements with the purpose of improving the performance of each element and thus of the entire node, and to make it possible to implement such a CCN node in reality with current hardware conditions.

Chapter 7

Future Works

This thesis addressed the question of improving the Content-Centric Network node performance and the implementation of such next generation network proposal within current technologies. The analysis and the evaluations of the three main contributions, described in this thesis, focus more on the performance of each individual CCN node. In order to go further on these works, I have some directions to suggest.

- **The DiPIT system**. The false positive rate is a key factor of this solution. In this thesis I have performed some simulations based on a linear topology and a Geant topology, with a zipf distribution traffic. We can still run more simulations based on different networking topologies and different networking traffic of various use cases, to see how the false positive rate and the packet loss ratio will be. The distributed Bloom filter based system is initially designed for the CCN PIT, but actually it can also be used for the other implementations, for example the CCN FIB. So how and where that we can extend this design is research direction that we can carry on in the future.
- **The content-aware dynamic forwarding system**. The OSPFN proposal, as well as some other recently presented solutions that are based on OSPFN, have already announced the first release. We can implement the content-aware forwarding system into the latest CCNx release or the CCN-lite [WWWb] which is developed by University of Basel, to compare the performance with the OSPNF protocol, in terms of networking traffic, memory requirement and the Data response time improvement. This content-aware forwarding design is not limited at the CCN usage. In the future we can also develop and adapt it into the other ICN networking solutions.
- **The interconnection of CCN and CDN**. This work is still at the early stage. In the next step we can also implement the cRouter into the CCNx or CCN-lite release, and use a network simulator, for example ns3, to run several simulations with the CDN server models to evaluate the CDN redirection ratio, packet loss ratio and the memory cost performances.

Finally, as I mentioned in the Chapter 6, the three contributions are coherent. After we valid each contribution independently through a vast evaluations, we can combine the them together and propose a whole CCN node that is based on these three optimized elements. After that we can still evaluate the performance by running some simulations or evaluations with different use cases, for example the P2P system, the online social networking use case, the WSN environment, or in an Internet of Things usage, to see how it performs in different conditions.

Bibliography

[ADI$^+$12] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A survey of information-centric networking. *Communications Magazine, IEEE*, vol.50, July 2012. 61

[ANO10] Somaya Arianfar, Pekka Nikander, and Jörg Ott. On content-centric router design and implications. In *ACM CoNext Workshop ReARCH*, 2010. 20, 23, 24

[BCA$^+$12] M.F. Bari, S. Chowdhury, R. Ahmed, R. Boutaba, and B. Mathieu. A survey of naming and routing in information-centric networks. *Communications Magazine, IEEE*, 50:44–53, 2012. 50

[BM04] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–509, 2004. 37

[BPC$^+$07] Paolo Baronti, Prashant Pillai, Vince WC Chook, Stefano Chessa, Alberto Gotta, and Y Fun Hu. Wireless sensor networks: A survey on the state of the art and the 802.15. 4 and zigbee standards. *Computer communications*, 30(7):1655–1695, 2007. 2

[BSAS13] Athula Balachandran, Vyas Sekar, Aditya Akella, and Srinivasan Seshan. Analyzing the potential benefits of cdn augmentation strategies for internet video workloads. In *Proceedings of the 2013 conference on Internet measurement conference, IMC2013*, pages 43–56, October 2013. 51

[CGMP13] Giovanna Carofiglio, Massimo Gallo, Luca Muscariello, and Michele Papalini. Multipath congestion control in content-centric networks. In *IEEE INFOCOM NOMEN workshop, Turin, Italy*, April 2013. 50

[CGP11] G. Carofiglio, V. Gehlen, and D. Perino. Experimental evaluation of memory management in content-centric networking. In *Proc. of IEEE ICC*, 2011. 24

[Cis] Cisco visual networking index: Forecast and methodology, 2012-2017. xiv, 2

[Con] ANR CONNECT research project. xvi

[DD11] A. Dhamdhere and C. Dovrolis. Twelve years in the evolution of the internet ecosystem. *IEEE/ACM Trans. Netw.*, 19(5):1420–1433, 2011. 63

[DKT03] Sarang Dharmapurikar, Praveen Krishnamurthy, and David E. Taylor. Longest prefix matching using bloom filters. In *ACM Sigcomm*, 2003. 25

[E$^+$11] Donald Eastlake et al. Rfc6195: Domain name system (dns) iana considerations. 2011. xiv, 2

[eri12] Ericsson mobility report: on the pulse of the networked society. Ercisson, Nov. 2012. `http://www.ericsson.com/news/1659597`. 61

[FRRS12] C. Fricker, P. Robert, J. Roberts, and N. Sbihi. Impact of traffic mix on caching performance in a content-centric network. In *IEEE INFOCOM NOMEN Workshops*, 2012. 56, 77

[GFMM12] V. Gehlen, A. Finamore, M. Mellia, and M. Munafo. Uncovering the big players of the web. In *Traffic Monitoring and Analysis*. Springer Berlin Heidelberg, 2012. 74

[GHM12] F. Guillemin, T. Houdoin, and S. Moteau. Statistics of youtube traffic in orange ip networks. Orange Labs draft, 2012. 51

[GKR$^+$11a] Ali Ghodsi, Teemu Koponen, Barath Raghavan, Scott Shenker, Ankit Singla, and James Wilcox. Information-centric networking: Seeing the forest for the trees. In *Proc. of HotNet'X*, 2011. 23

[GKR$^+$11b] Ali Ghodsi, Teemu Koponen, Jarno Rajahalme, Pasi Sarolahti, and Scott Shenker. Naming in content-oriented architectures. In *ACM Sigcomm Workshop ICN*, 2011. 24

[Gol10] D. Golding. The real story behind comcast-level 3 battle. GigaOM, Dec. 2010. `http://gigaom.com/2010/12/01/comcast-level-3-battle/`. 64

[GWCL06] D. Guo, J. Wu, H. Chen, and X. Luo. Theory and network applications of dynamic bloom filters. In *IEEE INFOCOM*, 2006. 37

[HAA$^+$13] A K M Mahmudul Hoque, Syed Obaid Amin, Adam Alyyan, Beichuan Zhang, Lixia Zhang, and Lan Wang. Nisr: named-data link state routing protocol. In *3rd ACM SIGCOMM workshop on Information-centric networking*, August 2013. 50

[HAM11] H. Hwang, S. Ata, and M. Murata. Realization of name lookup table in routers towards content-centric networks. In *Proc. of CNSM*, 2011. 23, 24

[JF08] Zbigniew Jerzak and Christof Fetzer. Bloom filter based routing for content-based publish/subscribe. In *ACM DEBS*, 2008. 25

[JST+09] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N Briggs, and R Braynard. Networking named content. In *CoNEXT '09*. ACM, 2009. xvi, 5, 13, 23, 27, 62

[JZER+09] Petri Jokela, András Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander. Lipsin: line speed publish/subscribe inter-networking. In *ACM Sigcomm*, 2009. 25

[KA98] Stephen Kent and Randall Atkinson. Rfc2401: Security architecture for the internet protocol, November, 1998. xiv, 2

[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. *SIGCOMM Comput. Commun. Rev.*, 37(4):181–192, August 2007. xvi, 5, 11, 24

[KMFB04] T. Karagiannis, M. Molle, M. Faloutsos, and A. Broido. A nonstationary poisson view of internet traffic. In *IEEE INFOCOM*, 2004. 39

[lcd12] Orange and akamai form content delivery strategic alliance. Akamai Press Release, Nov. 2012. `http://www.akamai.com/html/about/press/releases/2012/press_112012_1.html`. 64

[LKN13] Viktor Leis, Alfons Kemper, and Thomas Neumann. The adaptive radix tree: Artful indexing for main-memory databases. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, April 2013. 25

[LRH10] Uichin Lee, Ivica Rimac, and Volker Hilt. Greening the internet with content-centric networking. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 179–182, New York, NY, USA, 2010. ACM. 20

[LS10] Wencheng Lu and Sartaj Sahni. Low-power tcams for very large forwarding tables. *IEEE/ACM Trans. Netw.*, 18(3):948–959, June 2010. 25

[LS11] Zhe Li and Gwendal Simon. Time-shifted tv in content centric networks: The case for cooperative in-network caching. In *IEEE ICC*, 2011. 18

[LS13] Zhe Li and Gwendal Simon. In telco-cdn, pushing content makes sense. *IEEE Trans. on Networks and Services Management*, 2013. 64

[MA10] V. Vercellone et al. M.D. Ambrosio, M. Marchisio. Second netinf architecture description. In *Deliverable D6.2 of the 4WARD project*, Jan.2010. xvi, 5, 9

[MCG11] Luca Muscariello, Giovanna Carofiglio, and Massimo Gallo. Bandwidth and storage sharing performance in information centric networking. In *Proc. of the ACM SIGCOMM workshop ICN*, 2011. 23, 24

[MCL+11] Richard T. B. Ma, Dah Ming Chiu, John C. S. Lui, Vishal Misra, and Dan Rubenstein. On cooperative settlement between content, transit, and eyeball internet service providers. *IEEE/ACM Trans. Netw.*, 19(3):802–815, 2011. 63

[Moy98] John Moy. Rfc2328: Ospf version 2. April 1998. 17

[MPH+12] D. Migault, D. Palomares, E. Herbert, W. You, G. Ganne, G. Arfaoui, and M. Laurent. E2e: An optimized ipsec architecture for secure and fast offload. In *The seventh ARES*, 2012. 2

[MTP+11] Bertrand Mathieu, Patrick Truong, Jean-François Peltier, You Wei, and Gwendal Simon. *Information-centric networking: current research activities and challenges*, chapter Media Networks: Architectures, Applications and Standards. CRC Press, 2011. xiv, 3

[NB09] Erik Nordmark and Marcelo Bagnulo. Shim6: Level 3 multihoming shim protocol for ipv6. Technical report, RFC 5533, June, 2009. 2

[Ope12] Netflix Open Connect Peering Guidelines, 2012. `https://signup.netflix.com/openconnect`. 62

[ora12] Decision relating to practices concerning reciprocal interconnection services in the area of internet connectivity. Decision 12-D-18, Sep. 2012. French Competition Authority. 64

[PUR12] PURSUIT. FP7-INFSO-ICT-257217. Technical report, PURSUIT, April 2012. 13

[PV11] D. Perino and M. Varvello. A reality check for content centric networking. In *ACM Sigcomm workshop on ICN*, 2011. 20, 23, 24, 25, 28, 37, 55, 65, 74, 75

[QLC11] Yan Qiao, Tao Li, and Shigang Chen. One memory access bloom filters and their generalization. In *IEEE INFOCOM*, 2011. 37

[Ray09] D. Rayburn. More isps not letting cdn place servers inside their network, doing it themselves. `http://is.gd/UxExv6`, April 2009. 64

[RM04] V.C. Ravikumar and R.N. Mahapatra. TCAM architecture for IP lookup using prefix properties. *IEEE Micro*, 24(2):60–69, 2004. 25

[RR12a] D. Rossi and G. Rossini. On sizing ccn content stores by exploiting topological information. In *Proc. of IEEE Infocom NOMEN Workshop*, 2012. 20, 55

[RR12b] Giuseppe Rossini and Dario Rossi. A dive into the caching performance of content centric networking. In *IEEE CAMAD*, 2012. 61

[SAI13] SAIL. FP7-ICT-2009-5-257448/D-2.4. Technical report, Sail, Febreury 2013. 10

[SF02] Detlef Schoder and Kai Fischbach. Peer-to-peer. *Wirtschaftsinformatik*, 44, 2002. xiv, 3

[SHKL09] Haoyu Song, Fang Hao, M. Kodialam, and T.V. Lakshman. Ipv6 lookups using distributed and load balanced bloom filters for 100gbps core router line cards. In *IEEE INFOCOM*, 2009. 25

[SKBT12] D. Saucez, A. Kalla, C. Barakat, and T. Turletti. Minimizing bandwidth on peering links with deflection in named data networking. In *INRIA paper*, March 2012. 74

[Skl91] Keith Sklower. A tree-based packet routing table for berkeley unix. In *USENIX Winter*, volume 1991, pages 93–99, 1991. 25

[SNS+12] E. Suyong, K. Nakauchi, Y. Shoji, N. Nishinaga, and M. Murata. Catt: Cache aware target identification for icn. *Communications Magazine, IEEE*, vol.50, December 2012. 62

[SSRV11] Shashank Shanbhag, Nico Schwan, Ivica Rimac, and Matteo Varvello. Soccer: Services over content-centric routing. In *1st ACM SIGCOMM Information-Centric Networking (ICN) workshop*, August 2011. 50

[SX01] Randall R Stewart and Qiaobing Xie. Stream control transmission protocol (sctp). 2001. 2

[TL11] Rujiroj Tiengtavat and Wei-Ming Lin. Hybrid key duplication hashing techniques for ip address lookup. *INTERNATIONAL JOURNAL OF COMPUTER NETWORKS AND SECURITY*, 4(5):323–334, 2011. 25

[Tsi00] George Tsirtsis. Network address translation-protocol translation (nat-pt). *Network*, 2000. xiii, 2

[VdMSK02] Jacobus Van der Merwe, Subhabrata Sen, and Charles Kalmanek. Streaming video traffic: Characterization and network impact. In *Proceedings of the 7th International Web Content Caching and Distribution Workshop, IWCW*, August 2002. 51

[VP03] A. Vakali and G. Pallis. Content delivery networks: status and trends. *Internet Computing, IEEE*, 7(6):68–74, 2003. xiv, 3

[WHY+12] L. Wang, M. A. Hoque, Ch. Yi, A. Alyyan, and B.C. Zhang. OSPFN: An OSPF Based Routing Protocol for Named Data Networking. Technical Report NDN-003, Name Data Networking, July 2012. 17, 50, 64, 66

[Wil02] Carey L. Williamson. On filter effects in web caching hierarchies. *ACM Trans. Internet Techn.*, 2(1):47–77, 2002. 61

[WPD+10] M. Wittie, V. Pejovic, L. Deek, K. Almeroth, and B. Zhao. Exploiting locality of interest in online social networks. In *ACM CoNEXT 2010*, Novemver 2010. 51

[WRN$^+$13] Yaogong Wang, Natalya Rozhnova, Ashok Narayanan, David Oran, and Injong Rhee. An improved hop-by-hop interest shaper for congestion control in named data networking. In *3rd ACM SIGCOMM workshop on Information-centric networking*, August 2013. 50

[WVS$^+$99] Alec Wolman, Geoffrey M. Voelker, Nitin Sharma, Neal Cardwell, Anna R. Karlin, and Henry M. Levy. On the scale and performance of cooperative web proxy caching. In *ACM SOSP*, pages 16–31, 1999. 61

[WVTP97] Marcel Waldvogel, George Varghese, Jon Turner, and Bernhard Plattner. Scalable high speed ip routing lookups. In *ACM Sigcomm*, 1997. 25

[WWr] CCNx Repository. http://www.ccnx.org/releases/latest/doc/technical/RepoProtocol.html. 64

[WWWa] Anr CONNECT project. http://anr-connect.org. 13

[WWWb] CCN-lite. http://www.ccn-lite.net/. 85

[WWWc] Fp7 COAST project. http://www.coast-fp7.eu. 13

[WWWd] Network of information. http://www.netinf.org. 9

[WWWe] Project CCNx. http://www.ccnx.org/. 27, 64

[WWWf] Psirp. http://www.psirp.org. 12

[WWWg] Pursuit research project. http://www.fp7-pursuit.eu/PursuitWeb/. 13

[WWWh] Sail research project. http://www.sail-project.eu/. 10

[WZB13] Jason Min Wang, Jun Zhang, and Brahim Bensaou. Intra-as cooperative caching for content-centric networks. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, ICN '13, August 2013. 50

[YFR09] Minlan Yu, Alex Fabrikant, and Jennifer Rexford. Buffalo: bloom filter forwarding architecture for large organizations. In *ACM CoNEXT*, 2009. 25, 37

[YMB09] Heeyeol Yu, R. Mahapatra, and Laxmi Bhuyan. A hash-based scalable ip lookup using bloom and fingerprint filters. In *17th IEEE International Conference on Network Protocols, ICNP 2009*, October 2009. 25

[YMT$^+$12] W. You, B. Mathieu, P. Truong, J. Peltier, and G. Simon. Dipit: A distributed bloom-filter based pit table for ccn nodes. In *IEEE ICCCN*, August 2012. 55, 74

[ZEB$^+$10] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, KC. Claffy, D. Krioukov, D. Massey, C. Papadopoulos, T. Abdelzaher, L. Wang, P. Crowley, and Edmund

Yeh. Named data networking (NDN) project. In *Technic Report NDN-0001, Xerox Palo Alto Research Center-PARC*, October 2010. xvi, 13, 17

[ZGR+10] András Zahemszky, Borislava Gajic, Christian Esteve Rothenberg, Christopher Reason, Dirk Trossen, Dmitrij Lagutin, Janne Tuononen, and Konstantinos Katsaros. Experimentally-driven research in publish/subscribe information-centric inter-networking. In *Proc. of TridentCom*, 2010. xvi, 5, 12, 24

Presentations and Publications

Book Chapter

[1] Bertrand Mathieu, Patrick Truong, Jean-François Peltier, Wei You and Gwendal Simon. " Information-Centric Networking: Current Research Activities and Challenges" in *Media Networks: Architectures, Applications and Standards*, CRC Press, ISBN 978-1-4665-6658-3, 2011.

Conferences

[1] Wei You, Bertrand Mathieu, Patrick Truong, Jean-François Peltier and Gwendal Simon, " DiPIT: a Distributed Bloom-Filter based PIT Table for CCN Nodes", in *IEEE Int. Conf. on Computers, Communications and Networks (ICCCN)'2012*, 20 July - 2 August, Munich, Germany, 2012.

[2] Wei You, Bertrand Mathieu, Patrick Truong, Jean-François Peltier and Gwendal Simon. " Realistic Storage of Pending Requests in Content-Centric Network Routers", in *IEEE Int. Conf. on Communications in China (ICCC)'2012*, 15 - 17 August, Beijing, China, 2012.

[3] Wei You, Bertrand Mathieu and Gwendal Simon. " How to make Content-Centric Networks interwork with CDN networks", in *IEEE Network of the Future (NoF)'2013*, 23 - 25 October, Pohang, South Korea, 2013.

[4] Wei You, Bertrand Mathieu and Gwendal Simon. " Exploiting end-users caching capacities to improve Content-Centric Networking delivery", in *IEEE Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)'2013*, 28 - 30 October, Compiegne, France, 2013.

Journals

[1] Bertrand Mathieu, Patrick Truong, Wei You, and Jean-François Peltier, " Information-centric networking: A natural design for social network applications ", in *IEEE Communications Magazine*, July 2012, vol. 50, n. 7, pp. 44-51.

Patents

[1] Wei You, Bertrand Mathieu, Patrick Truong, and Jean-François Peltier, “ Internal architecture of CCN node to improve the management of message transfer”, No. 11 60632, 2012.

[2] Wei You, and Bertrand Mathieu, “ A Content-aware dynamic FIB table based on incoming content information for ICN networking”, No. 12 62260, 2012.

[3] Wei You, Bertrand Mathieu, and Gwendal Simon, “ New architecture in a CCN node for integration of repositories in a CDN network”, No. 13 52113, 2013.

Glossary

–A–	
AS	Autonomous System
ANR	Agence Nationale de la Recherche
–B–	
BGP	Border Gateway Protocol
BF	Bloom Filter
BO	Bit-level Object
BUFFALO	Bloom Filter Forwarding Architecture for Large Organizations
–C–	
c-a	content-aware
CCN	Content Centric Network
CDN	Content Delivery Network
COMET	COntent Mediator architecture for content-aware nETworks
CPU	Central Processing Unit
CR	Content Router
cRouter	cache Router
CS	Content Store
–D–	
DASH	Dynamic Adaptive Streaming over HTTP
DHT	Distributed Hash Table
DIFT	Dynamic Interest Forwarding Table
DiPIT	Distributed Pending Interest Table
DNS	Domain Name System
DO	Data Object
DONA	Data-Oriented Network Architecture
DRAM	Dynamic Random Access Memory
–F–	
FIB	Forwarding Information Base
FIFO	First In First Out
FN	Forwarding Node
fp	false positive
FTP	File Transfer Protocol

–H–	
HD	High Definition
HTCP	Hyper Text Caching Protocol
HTTP	Hypertext Transfer Protocol
–I–	
ICN	Information Centric Network
ID	Identifier
IETF	Internet Engineering Task Force
IO	Information Object
IP	Internet Protocol
IPsec	IP Security protocol
IS-IS	Intermediate System To Intermediate System
ISP	Internet Service Provider
–L–	
LFU	Least Frequently Used
LRU	Least Recently Used
LRFU	Least Recently/Frequently Used
LSA	Link State Advertisement
LSBD	Link State Data Base
–M–	
MDHT	Multiple Distributed Hash Table
–N–	
NAT	Network Address Translation
NetInf	Networking of Information
NRS	Naming Resolution System
–O–	
OLSA	Opaque Link State Advertisement
OSPF	Open Shortest Path First
OSPFN	OSPF for Named-data
–P–	
P2P	Peer to Peer
PC	Personal Computer
PIT	Pending Interest Table
PoP	Point of Presence
PRIT	Pending Repository Interest Table
PSIRP	Publish-Subscribe Internet Routing Paradigm
PURSUIT	Publish Subscribe Internet Technology
–Q–	
QoS	Quality of Service
QoE	Quality of Experience

–R–	
RAM	Random-Access Memory
RFT	Repository Forwarding Table
RH	Resolution Handle
RI	RendezVous Interconnection
RId	Resource Identifier
RLDRAM	Reduced-latency Dynamic random access memory
RN	RendezVous Node
RST	Reset
RTP	Real-time Transport Protocol
RTT	Round-Trip Time
–S–	
SAIL	Scalable and Adaptive Internet Solutions
SBF	Shared Bloom Filter
SDN	Software Defined Networking
SHA	Secure Hash Algorithm
SId	Scope Identifier
SRAM	Static Random Access Memory
SSD	Solid-state drive
–T–	
TCAM	Ternary content-addressable memory
TCP	Transmission Control Protocol
TN	Topology Node
TRIAD	Translating Relaying Internet Architecture integrating Active Directories
TTL	Time To Live
TV	Television
–U–	
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
–V–	
VoD	Video on Demand
–W–	
WSN	Wireless Sensor Network