ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2013-ENST-00xx

EDITE - ED 130 **Doctorat ParisTech**

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

«Spécialité: Informatique et Réseaux »

présentée et soutenue publiquement par

Giuseppe Rossini

le 31 Janvier 2014

Design Analysis of Forwarding Strategies for Host and Content Centric Networking

Directeur de thèse: **Dario Rossi**

Jury

M. James Kurose, Professor, University of Massachusetts	Rapporteur
M. Giacomo Morabito, Professore associato, University of Catania	Rapporteur
M. Serge Fdida, Professeur, Université Pierre et Marie Curie	Examinateur
M. Daniel Kofman, Professeur, Telecom ParisTech	Examinateur
M. Fabio Martignon, Professeur, Université Paris-Sud	Examinateur
M. James Roberts, Senior Researcher, INRIA	Examinateur
Mme. Giovanna Carofiglio, Head of Research Department, Bell Labs, Alcatel-Lucen	Examinateur

TELECOM ParisTech
école de l'Institut Mines-Télécom - membre de ParisTech
46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - www.telecom-paristech.fr

THÈSE

To my parents and to my sister.
And to you, my life partner, Dafne.

Acknowledgments

Thanking all the people who helped (voluntarily or not) during these three years would take at least another thesis. But this adventure has been magic. And even though one is the protagonist of its life, this tale includes other characters, teachers, magicians and friends, which I need to thank with all of my heart.

I really thank Dario for showing me the way, for the scoldings and the patience with my exuberant and distract personality. As a nice anecdote let me recall when I abruptly tried to move the server in a room without plugs and electricity! Thanks again Dario, I could not ask for better.

Thanks to my colleagues and friends: Silvio, Paolo, Claudio, Max, Stefano, Mattia, Andrea, Xavier, Thomas, Amy, Sameh, Yixi, Raffaele, Chiara, Leonardo, Claudio, Gege, Jessica, Marianna, Davide, Rosa, Matteo, Pina, Davide, Sharon. The list would surely be longer, and I apologize if I missed someone. However, even with an advice, with a joke or with a chat, all of you made me feel like home for these three years.

Thanks also to Giovanni, for his company, and for his delicious dishes that bring me back to Napoli.

Grazie alla mia famiglia. Ai miei zii, ai miei cugini, a mia nonna (con la quale, nonostante abbia oltre 80 anni, ho delle discussioni animatissime sull'Italia e sull'estero). Grazie anche al ramo allargato della famiglia: ai miei suoceri, a Stefano, ed agli zii Vastesi.

In particolare grazie ai miei genitori, che mi hanno aiutato a cercar casa, mi hanno aiutato a metterla a posto, e mi hanno offerto sempre ottimi consigli. Anche a distanza, anche se non a casa, sono sempre più convinto che avere una famiglia così accanto rappresenti davvero una marcia in più. Un punto di riferimento inamovibile.

Grazie anche a mia sorella Daniela. Anche se a volte posso nasconderlo le voglio un bene dell'anima, e la ringrazio per le sue burrascose visite delle quali ho un ricordo magico.

Infine, un ringraziamento speciale a te, Dafne. Grazie per innumerevoli ragioni, che prenderebbero anche queste pagine e pagine. Grazie per le nostre infinite risate, per sopportare la mia infinita sbadataggine e distrazione, grazie anche per le sgridate, per i litigi, per le riappacificazioni. Tu che sai farmi sentire protetto, e che allo stesso tempo necessiti della mia protezione. Che sai essere mamma e principessa allo stesso tempo. Insieme a te mi sento me stesso, completo. Quindi, grazie davvero per tutto, per essere così come sei, e per farmi essere così come sono. Ti amo. Sposami.

En fin, quelque mot en français (juste pour montrer d'être vécu trois ans et demi en France). Merci Paris, oui merci pour m'avoir accueilli dans tes bras, et pour m'avoir fait vivre des moments inoubliables. Les promenades, les restaurants, les concerts, les fêtes. J'espère de te rencontrer, peut-être, un jour. Merci.

Abstract

Starting from the evidence of the Internet's actual limits, in this Thesis we investigate different aspects of two directions the Internet is evolving toward. In particular, we consider more flexible ways to reach hosts, and to distribute content.

Host Centric Networking (HCN) is the name we give to the umbrella architectures which try to decouple host location and identifiers. Basically, they identify each device by the means of flat labels which do not locate the host within the network.

HCN architectures leverage Distributed Hash Table(DHT) approaches for retrieving the host position from the corresponding label. However, routing and forwarding underlying the DHT, heavily rely on traditional single path algorithms. Thus, in the first part we propose Adaptive Probabilistic Link-state for Switching In Autoforwarding(Aplasia), an alternative routing architecture mainly composed by a path-finding algorithm, namely Adaptive Probabilistic Link-state(Apl), and by an autoforwarding data plane.

By adding a slight amount of message complexity to the canonical approaches, Apl is able to find optimal disjoint paths, still reducing computational algorithmic complexity . The trade-off between communication cost and path optimality is tuned by a single system parameter, easily set and by the way not critical for the correct Aplasia functioning. Besides, we provide a model which forecasts static and dynamic costs of our approach.

Aplasia leverages an autoforwarding data plane that completely specifies paths in the packet header. This kind of design makes removing FIBs from the network core possible, following the impetus towards simplifying core devices, and shifting complexity to the edge. Finally, we evaluate Aplasia's autoforwarding by the means of a Click testbed.

Information Centric Networking (ICN) makes content directly addressable by network hosts. The basic idea is to send packets carrying the content identifier, rather than the host address. As content can be easily cached within network devices, an ICN network can be modeled as a receiver driven network of caches.

Indeed, in the second part of this work, we consider caching algorithms deployed over a network of caches. Each of these algorithms is a triplet composed by forwarding (which path is worth following), meta-caching (what content is worth caching), and replacement (what content is worth replacing) strategies.

We develop ccn$\mathcal{S}$im (distributed like open source software) in order to inspect which (exogenous and endogenous) factors mostly influence caching performance: popularity models, topologies, strategies, and henceforth. Then, we focus on the forwarding part, pointing out that coupling meta-caching and forwarding strategies produces a notable performance gain. We then propose a theoretical model for an ideal forwarding strategy, in which the nearest copy of the content is instantaneously provided by an external oracle. Finally, we design two different forwarding implementations of the oracle-based approach.

Résumé

À partir des limites évidentes d'Internet, dans cette thèse nous étudions différents aspects des deux directions vers lesquelles l'Internet est en train d'évoluer. En particulier, nous considérons des moyens plus flexibles pour joindre les hôtes, ou pour distribuer le contenu dans la réseau.

L'*Host Centric Networking* (HCN) est le nom que nous donnons à l'ensemble des architectures qui tentent de découpler la position et l'identification d'un hôte. En fait, les architectures HCN identifient chaque nœud par des étiquettes plates qui ne localisent pas l'hôte dans le réseau.

Les architectures HCN utilisent des *Distributed Hash Tables* (DHT) pour récupérer la position de l'hôte à partir de l'étiquette correspondante. Toutefois, l'acheminement et la transmission sous-jacente à la DHT s'appuient fortement sur des algorithmes traditionnels basés sur des chemins uniques. Ainsi, dans la première partie de cette Thèse, nous proposons *Adaptive Probabilisti Link-state for Switching In Autoforwarding*(APLASIA), une architecture de routage alternative composée principalement par un algorithme de recherche des chemins, à savoir Adaptive Probabilistic Link-state(APL), et par un plan de données de type *autoforwarding*.

En ajoutant une petite quantité des messages à la complexité des approches canoniques, APL est capable de trouver des chemins disjoints optimaux, en réduisant au même temps la complexité computationnelle. Le compromis entre le coût de communication et l'optimalité des chemins peut être réglé par un paramètre de système simple à fixer et au même temps pas critique pour le fonctionnement d'APLASIA. Par ailleurs, nous fournissons un modèle qui peut prédire les coûts statiques et dynamiques de notre approche.

APLASIA s'appuie sur un plan des données *autoforwarding* qui spécifie complètement les chemins dans l'en-tête du paquet. Ce type de conception permet la suppression des FIBs dans le cœur du réseau, en simplifiant les routeurs du cœur, et en déplaçant la complexité vers la périphérie du réseau. Enfin, nous évaluons cette proposition par un banc d'essai *Click*.

L'*Information Centric Networking* (ICN) rend le contenu directement adressable par les hôtes du réseau. L'idée de base consiste à envoyer des paquets portant l'identifiant

du contenu, plutôt que l'adresse de l'hôte. Puisque le contenu peut être facilement stocké dans une mémoire cache, un réseau ICN peut être modélisé comme un réseau de mémoires caches dirigé par le clients.

Dans la deuxième partie de ce travail, nous considérons différents algorithmes déployés sur un réseau de mémoires caches. Chaque algorithme sera constitué par une stratégie d'acheminement (sélection de chemin), une stratégie de *meta-caching*(pour décider s'il faut stocker le contenu), et une stratégie de remplacement (pour décider quel contenu il faut replacer dans la mémoire cache).

Nous développons ccn$\mathcal{S}$im (distribué sous forme d'un logiciel *Open Source*) afin de vérifier quels facteurs influencent principalement les performances des algorithmes ci-dessus: les modèles de popularité, les topologies ou la choix des différentes stratégies. Puis, nous nous concentrons sur la partie d'acheminement, en soulignant que le couplage entre le *méta-caching* et les stratégies d'acheminement améliore remarquablement les performances. Nous proposons ensuite un modèle théorique pour une stratégie idéale d'acheminement, dans laquelle la copie la plus proche du contenu est instantanément localisée par un oracle externe. Enfin, nous proposons deux différentes implémentations des stratégies basées sur l'oracle.

Synthèse en Français

Introduction

Les questions d'extensibilité affectent l'Internet depuis sa naissance. Pour résoudre ces problèmes, d'habitude ont été seulement présenté des simples *patches* au lieu de modifier le cœur de la pile de communication. Dans la Fig. 1 nous montrons comment la couche de réseau a été resté inchangée dans les derniers 20 ans, tandis que d'autres niveaux ont été étendus et parfois ajouté dans la pile. Ce phénomène a finalement mené à l'ossification d'Internet autour du protocole IP. Le problème de *l'ossification d'Internet* [20–22] provient de la simple observation que, en changeant le protocole IP, il faut forcement reconfigurer les routeurs du cœur du réseau. Au contraire, l'évolution des protocoles de plus haut niveau est plus simple, et peut être progressivement déployé. Cette simple observation pousse à développer des applications sur la couche TCP/IP qui imitent des réseaux réels, mais sont composées par connexions virtuelles, de façon que on est pas obligé à modifier les niveaux plus bas du réseau. Le large développement de boîtes intermédiaire HTTP/TCP (par exemple, les boites *NAT* ou pare-feu) a ultérieurement conduit à scléroser la pile autour de ces protocoles [22–24].

L'ossification d'Internet représente la racine de la plupart des maux qui sont de plus en plus graves dans le scénario de nos jours: la sécurité, la gestion des réseaux, la performance, la flexibilité en représentent qu'une petite partie. Même si la nécessité de résoudre ces problèmes semble toujours pas urgente, décrire les solutions possibles semble obligatoire pour la communauté scientifique. Ainsi, la recherche dans le domaine des architectures futures tente de simplifier et optimiser l'état actuel de l'Internet. L'objectif de haut niveau est d'emprunter des idées et des mécanismes à partir des solutions applicatives, en essayant de mettre ces mécanismes dans le cœur de l'Internet. En particulier, nous nous concentrons sur deux approches différents:

- Sous le nom de *Host Centric Networking*(HCN, ou Réseaux centrés sur les hôtes), nous classons les architectures de coordination qui mettent l'accent sur le découplage entre l'identification de l'hôte et son emplacement. Dans ce façon, on essaie de résoudre le problème de la extensibilité lié a l'expansion des tables du routage dans le cœur d'Internet [25].

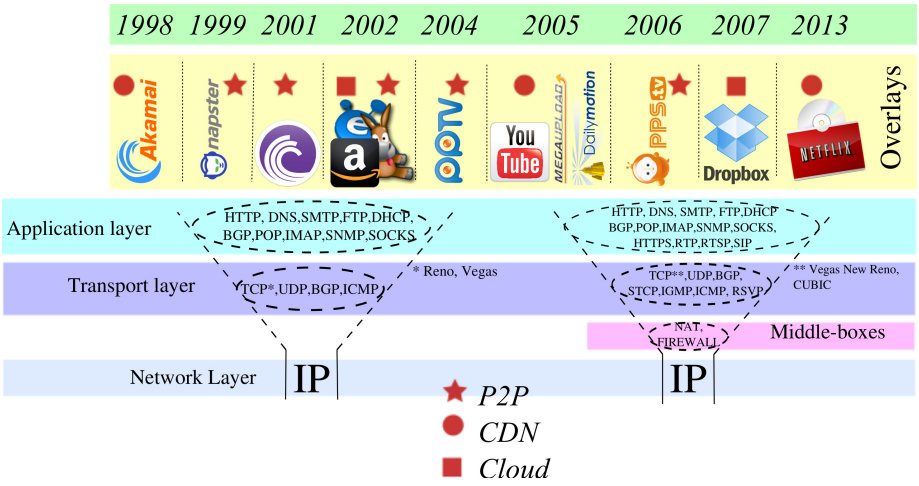


Figure 1: Représentation de l'évolution d'Internet dans les dernières années.

- Avec l'*Information Centric Networking* (ICN, ou Réseaux centrés sur les contenus), nous nous référons à l'ensemble des futures architectures Internet qui rendent le contenu directement adressable au sein de la couche réseau IP. ICN essaie de résoudre les problèmes de scalabilité liés à l'augmentation exponentielle du trafic Internet [26].

Host Centric Networking

Au cours des dernières années, plusieurs projets de recherche ont abordé la question du découplage du concept d'identification des hôtes et de ses emplacement, soit pour l'Intranet [27–30] ou, plus récemment, au niveau Internet [31–33]

L'idée de base derrière à un tel découplage est de distribuer l'état des routeurs dans une *Distributed Hash Table* (DHT, ou table de hachage distribuée). En fait, dans l'état actuel, IP emploie un adressage hiérarchique pour agréger hôtes différents sous la même sous-réseau. Toutefois, la hiérarchie exige noms dépendants de l'emplacement qui compliquent la gestion du réseau (par exemple, la mobilité) et qui donc compliquent la distribution des tables de routage. Les noms dépendants de l'emplacement peuvent fournir assez d'avantages pour devenir attrayant seulement pour les environnements spécialisés (par exemple, comme dans Portland [34], qui exploite une adressage basée sur la position pour optimiser l'acheminement dans le centre de traitement de données, ces derniers disposés dans une topologie de type *fat tree*). Pour les besoins plus généraux, noms indépendants de l'emplacement, aussi dit *plat*s, ont reçu un intérêt croissant dans ces derniers temps [33]. Les identifiants plats seront donc stockés dans la table de hachage distribuée, qui, une fois questionné, récupérera la position réelle de l'hôte dans le réseau.

Souvent, ces architectures utilisent des protocoles de routage traditionnels (par exemple

Table 1: Comparaison d'effort lié.

Architecture	**Résolution de l'hôte**	**Routage**	**Complexité algorithmique**	**Complexité de communication**	**Plusieurs chemins**
Rbridges [28]	Centralisée				Non
SEATTLE [30]	DHT Hop single	LS (OSPF)	$O(NlogN + N\delta)$	$O(N\delta)$	Non
ROFL [32]	DHT Chord				Non
SmartBridge [27]	Centralisée	Computation diffusée + BFS	$O(N\delta)$	$O(N\delta)$	Non
BANANAS [31]	n.a.	LS + [35]	$O(N\delta + NlogN + kN)$	$O(N\delta)$	Oui (k)
Viking [29]	n.a.	Centralisé + plusieurs exécutions de [35]	$O(N^3logN + N^3\delta)$	n.a.	Oui (2 sur k)
APLASIA	n.a.	APL	$O(N\delta)$	$O(N\delta)$	Oui (2)

OSPF ou IS-IS) pour acheminer soit les requêtes adressées à la DHT, soit les données réelles. Ces protocoles canoniques nous révèlent leurs limites lorsqu'ils sont utilisés dans des environnements plats. Notre objectif est de proposer une architecture pour le routage sur chemins multiples dans des réseaux plats, en ciblant particulièrement les routage intra-domaine. Notre proposition se trouve à un point radicale, inexplorée jusqu'ici, dans l'espace de conception de réseau, où nous gagnons de simplicité algorithmique, par rapport à une légère et réglable incrément du coût de communication. Nous appelons notre architecture de routage *Adaptive Probabilistic for Link-state Architecture Switching in Autoforwarding* (Aplasia) .

D'ailleurs, Aplasia décale l'état du routage dans l'en-tête des messages circulant dans le réseau, enlevant *de facto* les FIBs du cœur du réseau du Fournisseur d'Accès à l'Internet (FAI).

Nous rappelons que, dans ce travail, nous ne nous concentrons pas sur le problème de résolution de nom, qui peut être traité, par exemple, comme dans [30]. Au contraire, nous nous concentrons sur deux aspects du système: le routage et l'acheminement. Nous résumons et comparons les efforts pertinents dans ce domaine dans Tab. 1, en termes soit de complexité computationnelle, soit de complexité de communication (c.-a.-d., nombre de messages). Dans Tab. 1 nous énumérons les propositions de routage sur des identifiants plats, à partir d'une perspective Internet [31–33] ou Intranet [27–30, 36] (ce dernier étant plus proche de notre travail sur le routage intra-domaine).

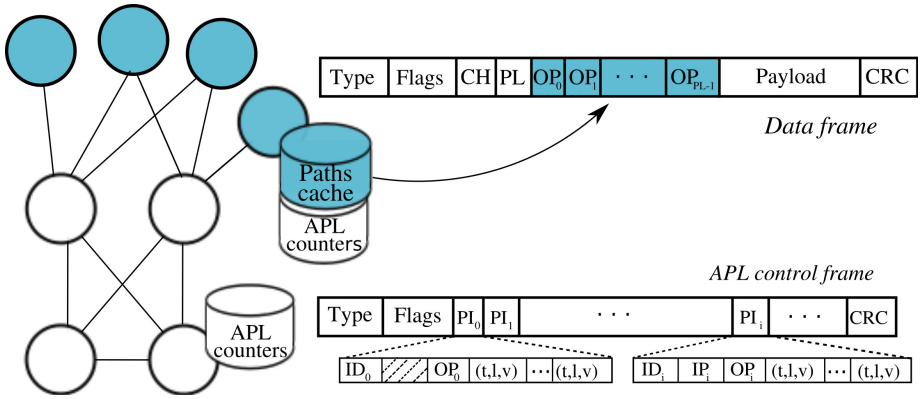


Figure 2: Synopsis d'Aplasia le plan de contrôle, et un sketch des trames des données et de contrôle.

Adaptive Probabilistic Link-state Architecture Switching in Autoforwarding

Nous décrivons notre proposition de routage, à savoir Aplasia. Les ingrédients principaux de notre recette sont (i) l'utilisation de l'algorithme gourmand de routage probabiliste *Adaptive Probabilistic Link-state* (Apl) pour découvrir rapidement plusieurs chemins, proche de l'optimum, dans le plan de contrôle et (ii) l'utilisation de chemins d'accès complets directement dans l'en-tête des trames, qui permet aux appareils d'acheminer les données sans consulter leur table de routage. Nous simulons, analysons et mettons en œuvre notre proposition afin de montrer sa solidité. Dans ce qui suive, on décrit le principaux composants d'Aplasia, montrées dans Fig. 2.

L'architecture des nœuds Les nœuds sont gérés par des identifiants plats qui sont leur associés de manière univoque (comme dans [30, 32, 33]), et peuvent être choisis indépendamment de la position topologique des nœuds. Chaque fois qu'un nœud veut envoyer des données vers un autre nœud, il a besoin d'assembler une trame, en précisant le chemin complète bout-à-bout dans l'en-tête de la trame. La trame est alors remise au plan de données pour le transfert. Puisque dans l'en-tête est spécifié eune séquence de chemin d'accès complet, la trame est tout simplement poussée à l'interface de sortie à laquelle fait référence le champ *Current Hop*(CH) (rappelons que $CH = 0$ lors de la création d'image, et est incrémenté de 1 à chaque saut). En outre, tous les nœuds au long du trajet effectuent la même opération de transfert, de sorte que pas de recherche est effectuée dans les tables de commutation. En gros, en référence à la Fig. 2, nous pouvons identifier deux types de nœuds dans le réseau, à savoir nœuds de bords et nœuds cœur: les premiers peuvent être des sources de trafic, alors que les deuxièmes n'effectuent pas de commutation fonctionnalités. Aplasia

suit le principe de pousser la complexité vers la périphérie du réseau, de sorte que les nœuds de base doivent garder uniquement (i)un montant minimal d'état (les compteurs de APL , comme on verra) pour exécuter l'algorithme probabiliste adaptatif APL . Les nœuds de bord stockent un cache (ii) des parcours. C'est intéressant de souligner que les compteurs de l'algorithme APL sont utilisés dans le plan de contrôle uniquement pendant l'échange de messages de routage et que, de même, la mémoire cache contenant les chemins de routage est utilisé par les nœuds du bord seulement au moment de la génération de la trame. Donc, ces structures sont accessibles à une vitesse beaucoup plus lente par rapport aux opérations du plan d'acheminement des données dans le cœur du réseau.

Les trames de données Fournir des détails complets sur la spécification et l'encodage de la trame d'APLASIA est hors de la portée de ce résumé. Par contre, nous décrivons les principaux champs à l'aide de la figure Fig. 2, esquissant les trames de données et de contrôle. Mis à part les champs habituels tels que le type de trame, le ID, le drapeau, indications de qualité de service et de contrôle, les trames de données portent une liste des identifiants $\{OP_i\}_i$, qui représente l'ensemble des ports de sortie à suivre bord-à-bord dans le domaine APLASIA, avec un pointeur CH qui tient en compte l'interface suivante et un champ de longueur de chemin PL. Par défaut, chaque OP_i consomme 8 octets dans l'en-tête (optimisé pour les nœuds ayant au plus 256 ports). Ce choix a un certain nombre d'avantages, dont le premier est de simplifier le reste de l'architecture en produisant un acheminement sans état. Cette simplicité vient au prix d'un légère augment de la longueur de l'en-tête de trame, qui croît proportionnellement à la longueur du trajet.

Les trames de contrôle Les messages de contrôle contiennent des informations de chemin supplémentaires (PI). Toutefois, comme le volume des messages de contrôle est faible par rapport aux échanges de données, les frais concernant le plan de contrôle résultent être limitées. Dans le détail, PI_i contient, outre l'interface de sortie OP_i, l'identifiant de nœud correspondant ID_i (utilisé pour détecter des boucles) et les ports d'input IP_i (utilisés par le nœuds pour *inférer le chemin depuis la trame qu'ils voient passer*). La séquence PI croît à chaque saut pendant le processus de calcul de chemin. Des informations facultatives sur la qualité estimée du chemin peuvent être transportée dans l'en-tête, sous la forme de triplets *type-longueur-valeur* (t, l, v) en PI_i, pour faciliter les opérations d'ingénierie du trafic (hors de la portée de ce travail). Puisque les messages de contrôle portent des informations concernant le chemin, un nœud de bord, en gérant un paquet, peut déduire des informations topologiques de manière totalement passive: plus précisément, un nœud qui reçoit un message de contrôle qui a déjà parcouru i soutes, peut en principe apprendre (en remplissant le mémoire cache des chemins) les chemins des précédents $i - 1$ nœuds jusqu'à l'origine. Puisque les différents messages portent éventuellement des informations différentes, des chemins multiples vers la même destination sont effectivement trouvés.

Algorithme Adaptive Probabilistic Link-state (APL)

Dans cette section, nous analysons, modélisons, et évaluons, APL , l'algorithme probabiliste composant APLASIA. Considérons un réseau modélisé comme un graphe non orienté $G = (E, V)$, composé par $|V| = N$ routeurs. Entre chaque pair de routeurs $i, j \in V$, nous sommes intéressés à trouver un couple de chemins, c'est à dire des séquences nœuds qui connectent i à j. On appelle $\mathcal{P}_{i,j}$ et $\mathcal{S}_{i,j}$ les chemins primaires et secondaires, respectivement, fournis par APL sur le graphique G. On note $lenght(\mathcal{P}$)et $length(\mathcal{S}$)les longueurs respectives de ces chemins.

- Le chemin primaire $\mathcal{P}_{i,j}$ est défini comme le chemin le plus court trouvé qui connecte i et j. On dit que le chemin primaire est optimal si sa longueur est la même obtenu par un algorithme centralisé (comme Dijkstra).

- Le chemin secondaire $\mathcal{S}_{i,j}$ est défini comme le chemin le moins similaire à $\mathcal{P}_{i,j}$, déterminé par APL . Pour trouver le chemin secondaire optimal, nous considérons un graphe modifié G' dans lequel les coûts des liaisons au long du chemin primaire optimal sont augmentées par le diamètre du réseau [37], et les autres coûts sur le liens sont unitaires. En exécutant Dijkstra sur G', nous retrouvons un chemin $\mathcal{S'}_{i,j}$ minimisant la fonction de similarité $\mathcal{P'}_{i,j} \cap \mathcal{S'}_{i,j}$. Nous disons que le chemin secondaire $\mathcal{S}_{i,j}$ trouvé par APL est optimal si $|\mathcal{P}_{i,j} \cap \mathcal{S}_{i,j}| = |\mathcal{P'}_{i,j} \cap \mathcal{S'}_{i,j}|$ et $length(\mathcal{S}_{=})L_{S'_{i,j}}$, c'est à dire, la longueur $length(\mathcal{S}$)du chemin secondaire est égale à la longueur $L_{S'_{i,j}}$ de l'optimal $\mathcal{S'}_{i,j}$ (comme il peut y avoir plusieurs chemins disjoints minimisant la similitude avec le chemin le plus court).

Une description du pseudocode de l'algorithme est donnée dans Alg. 1. Un nœud de source s commence le processus d'annonce par l'inondation d'une paquet m à tous ses voisins. Le paquet inondé contient une liste d'identifiants ID, initialement fixé à $ID[0] = s$ par la source, qui ajoute à chaque nœud son propre identifiant. Lors de la réception d'un paquet d'annonce m, un nœud apprend le chemin (en arrière) vers la source s et à n'importe quel nœud intermédiaire $d = m.ID[i]$ au long du chemin. Dans le cas où le récepteur j détecte une boucle (en trouvant son identifiant dans la liste de ID), il ignore le message et annule la procédure d'inondation. Sinon, il l'analyse et éventuellement mémorise le chemin appris $\mathcal{O}_{j,d}$. Le chemin primaire (et secondaire) est d'abord stocké dans le routeur si il ne existe pas encore. De plus, si le chemin nouvellement entendu est plus court que le chemin principal $length(\mathcal{L}_{j,d}) < length(\mathcal{P}_{j,d})$, alors le chemin primaire est mis à jour avec celui entendu. De même, si le chevauchement entendu $|\mathcal{P}_{j,d} \cap \mathcal{L}_{j,d}| < |\mathcal{P}_{j,d} \cap \mathcal{S}_{j,d}|$ est inférieur au chemin secondaire courant, ou s'il a égale similitude mais est plus court que le secondaire $|\mathcal{P}_{j,d} \cap \mathcal{L}_{j,d}| = |\mathcal{P}_{j,d} \cap \mathcal{S}_{j,d}| \wedge length(\mathcal{L}_{j,d}) < length(\mathcal{S}_{j,d})$, alors le chemin secondaire est mis à jour.

Algorithm 1: Pseudocode de l'algorithme APL exécuté par un nœud j du réseau.

```
1  while j is receiving message m do
2      ℓ ← length(m.ID)
3      forall i ∈ [0, ℓ] do
4          if m.ID[i] = j then                                  // loop and abort flooding
5              return
6          else
7              d ← m.ID[i]                                                  // Destination
8              L_{j,d} ← (m.ID[ℓ], ..., m.ID[i])                       // Path overhearing
9              if ∄P_{j,d} ∨ length(L_{j,d}) < length(P_{j,d}) then
10                 P_{j,d} ← L_{j,d}                                // Update primary path
11             cond_1 = |P_{j,d} ∩ L_{j,d}| < |P_{j,d} ∩ S_{j,d}|
12             cond_2 = |P_{j,d} ∩ L_{j,d}| = |P_{j,d} ∧ length(L_{j,d}) < length(S_{j,d}))
13             if ∄S_{j,d} ∨ cond_1 ∨ cond_2 then
14                 S_{j,d} ← L_{j,d}                            // Update secondary path
15     append j to m.ID
16     s ← m.ID[0]                                                          // Source
17     forall next ∈ neighbors(j) do
18         if next ≠ m.ID[ℓ − 1] then
19             send m to next w.p. β^{n_s}                      // Probabilistic flooding
20     n_{j,s}++                                  // Update counter associated with source s
```

Le nœud j inonde un message de contrôle produit par le nœud source s sur tous ses liens (sauf celui dont il a reçu le message) avec la probabilité :

$$P = \beta^{n_{j,s}} \tag{1}$$

β est défini comme paramètre de *backoff* et $n_{j,s}$ est un compteur, stocké au nœud i, du nombre de fois que le nœud j a déjà reçu un paquet produite par le nœud s. Les décisions d'inondation sont prises indépendamment sur chaque lien et le compteur est remis périodiquement.

Complexité de l'algorithme probabiliste

Dans cette section, nous quantifions précisément le compromis entre les frais d' APL et la qualité du chemin. Nous évaluons d'abord les frais généraux de l'algorithme à travers un simple modèle analytique, et nous employons de simulation à évents discrets pour valider l'analyse et l'évaluation des performances en termes de qualité des chemins découverts.

Les simulations sont effectuées par Omnet++ [38] avec des topologies de réseau différentes, dont les propriétés sont résumées dans Tab. 2. Plus précisément, Tab. 2 indique le nombre de nœuds N , la moyenne et l'écart type du degré des nœuds, δ et σ, et le diamètre D du graphe G .

Network	Segment	N	δ	σ_δ	$\Delta[ms]$	D
Qwest	Cœur	33	5.0	3.1	5.9	5
DTelekom	Cœur	68	10.4	13.3	17.2	3
Level3	Cœur	46	11.7	10.1	8.9	4
Sprint	Cœur	315	6.2	6.9	3.2	13
Geant	Aggr	22	3.4	1.4	2.6	4
Tiger2	Métro	22	3.6	0.6	0.1	5
Random	-	$10^{[2,5]}$	4	≈ 4	1	$[3, 6]$

Table 2: Propriétés topologiques des scénarios des réseaux.

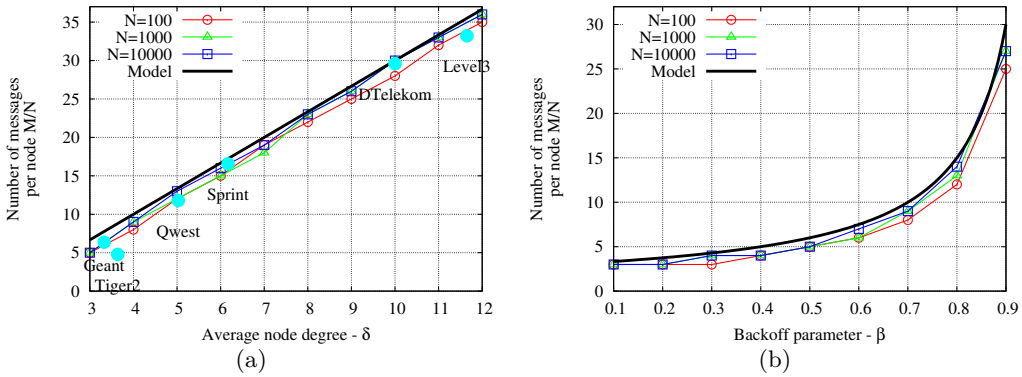


Figure 3: Comparaison entre modèle et simulation du nombre des messages par chaque nœud en variant le degré des nœuds δ, fixant $\beta = 0.7$ (a) et le paramètre de backoff β en fixant $\delta = 4$ (b).

Nous soulignons que nous considérons soit topologies FAI réelles, correspondantes à différents segments du réseau, ainsi que un ensemble de 50 graphes aléatoires synthétiques avec $N \leq 10000$ et $\delta = 4$. Pour l'instant, nous utilisons des paramètres homogènes (c.-à-d., un retard constant et égal sur chaque lien), et on assume qu'aucune panne se produise au sein du réseau.

Nous évaluons maintenant le coût de l'algorithme APL en termes de complexité de communication, d'espace et de calcul.

Complexité de Communication Le nombre total de messages de contrôle M transmis sur le réseau au cours d'une seule opération d'inondation peut être facilement estimé en négligeant la topologie réelle du réseau. Nous avons que si le nœud s a entamé le processus d'avertissement, depuis un nœud générique j départent $\delta - 1$ messages (c.-à-d. sur toutes les interfaces sauf l'interface à partir de laquelle le message est venu) avec une probabilité donnée dans Eq. (1) qui dépend du nombre de fois qui

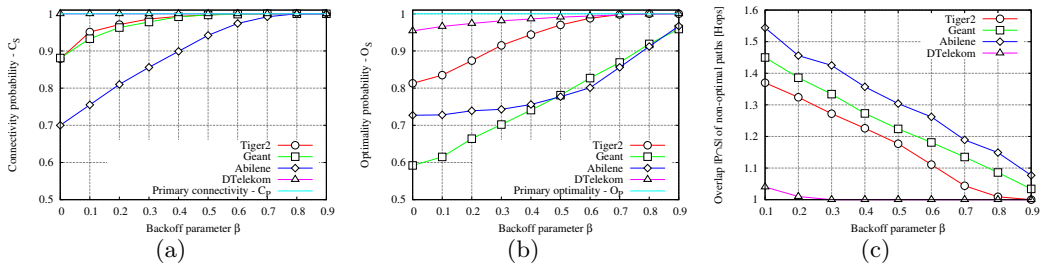


Figure 4: Qualité des parcours déterminé par APL . On montre la connectivité (a) l'optimalité (b) et le chevauchement entre le parcours primaire et secondaire (c) sur des topologies balistiques.

il a géré le message du même annonceur. Le nombre de messages totale du réseau peut être alors recueilli en additionnant simplement toutes les possibles valeurs du compteur $c_j \in [0, \infty]$, et considérant que tous les nœuds font le même en multipliant par N:

$$\begin{aligned} M &\approx N\left((\delta-1)+(\delta-1)\beta+(\delta-1)\beta^2+\cdots\right) \\ &= N(\delta-1)\sum_{n=0}^{\infty}\beta^n = N\frac{\delta-1}{1-\beta} \end{aligned} \tag{2}$$

Surtout, l'Eq. (2) montre que le nombre de messages M répartis sur le réseau pendant une avertissement dépend linéairement de la taille du réseau et du degré moyenne δ, et hyperboliquement de $1-\beta$. On rappelle que (N, δ) sont donnés par le scénario et ils ne peuvent pas être changés dans le déploiement réel. Par contre, le paramètre de *backoff* β donne un bouton très simple pour régler les frais généraux. Le nombre de messages généraux par une algorithme de type Link State (LS, ou à état de liens) est $\delta N(N-1)$. Donc, APL augment le coûts de communication d'un facteur $1/(1-\beta) > 1$, assurant que ils soient similaires avec ceux généraux par une algorithme LS.

Nous validons Eq. (2) par simulation en Fig. 3, où on normalise le nombre de messages sur la taille du réseau pour simplifier la comparaison sur des réseaux ayant des tailles hétérogènes. Nous simulons soit des vraies topologies considérées précédemment (cercles pleins, pour montrer la précision du modèle dans la pratique) ainsi que les graphes aléatoires avec des degrés variables $\delta \in [3, 12]$ et la taille $N \in \{100, 1000, 10000\}$ (cercles vides). La Fig. 3 indique le nombre de messages traités par nœud M/N pendant une rond d'inondation, en comparant le modèle et la simulation pour différents degrés

moyens (Fig. 3(a)) et différents valeurs de β (voyer la Fig. 3(b)), à partir duquel c'est possible recueillir dans les deux cas une excellente adaptation .

Complexité spatiale Le stockage nécessaire pour exécuter l'algorithme APL est $O(2N)$ afin de stocker l'arbre primaire et secondaire/backup du réseau (comme dans [29]). Nous avons également besoin de $O(N)$ compteurs $n_{j,s}$ pour compter sans risque le nombre de messages générés par chaque source. Cela pourrait être problématique dans le cas où la taille du réseau n'est pas connue à priori (toutefois N pourrait être fixé à un nombre assez grand pour assurer de ne pas avoir de chevauchements).

Complexité computationnelle Dijkstra et les autres algorithmes de graphes ont besoin d'effectuer $O(NlogN)$ et $O(NlogN + E^2)$ opérations pour le calcul du chemin le plus court et le chemin de backup, une fois que le graphe complet est connu. APL d'autre part doit effectuer des opérations simples, paquet par paquet. Plus précisément, à la réception de chaque message de contrôle, les routeurs doivent effectuer: (i) une comparaison de la trajectoire la plus courte (Alg. 1, ligne 9), (ii) une intersection pour le meilleur chemin alternatif (Alg. 1, ligne 12). Comme le nombre des messages de contrôle global d' APL est limité, nous pouvons lier la complexité de APL à l'Eq. (2). Il croit avec N^2 lorsque toutes les annonces commencent au même instant. Au lieu de cela, c'est intéressant remarquer que dans les algorithmes à état de liens (par exemple, OSPF ou IS-IS), un simple annonce d'état pour un changement de topologie, provoque, pour chaque nœud, une exécution de l'algorithme de Dijkstra avec une complexité $O(\log NN)$.

Performance de l'algorithme

Nous nous focalisons maintenant sur la qualité des chemins que l'algorithme d'affichage probabiliste adaptative est capable de trouver. Nous laissons chaque nœud annoncer lui-même une seule fois au temps $t = 0$ et évaluons la connectivité et l'optimalité des chemins primaires et secondaires. Puisque l'évaluation de la qualité du chemin des réseaux aléatoires n'est pas réaliste, nous ne considérons que des topologies FAI, en présentant des résultats obtenus avec plus de 20 simulations par topologie.

Nous exprimons la qualité du chemin en termes de connectivité au long du chemin primaire et secondaire (si les chemins $\mathcal{P}_{i,j}$ et $\mathcal{S}_{i,j}$ joindre les nœuds $i, j \in V$) et optimalité (si $\mathcal{P}_{i,j}$ et $\mathcal{S}_{i,j}$ sont optimales selon les définitions ci-dessus). Nous exprimons la connectivité en termes de probabilité C_P (C_S) que $\forall i, j \in V$ les nœuds i et j sont reliés par un parcours primaire (secondaire). Nous exprimons l'optimalité en termes de probabilité O_P que le chemin principal est aussi le plus court, et en termes de probabilité O_S que le parcours secondaire est le plus court chemin et, au même temps, le plus diversifié du primaire.

Fig. 4(b) reportes la *probabilité d'optimalité* des chemins primaires (le chemin le plus court) et secondaires (le plus diffèrent du chemin primaire) en fonction de β: puisque le chemin le plus court est toujours trouvé, l'optimalité du parcours primaire est garanti.

Ainsi, l'indice d'optimalité n'est significatif que pour le chemins secondaires: nous voyons qu'un pourcentage important (de 60 % à 85 % , en fonction de la topologie) des chemins secondaires sont optimales, même pour une très faible valeur de $\beta = 0.1$, et qu'au moins 90% de chemins secondaires sont optimales quand $\beta \geq 0.8$, pour toutes les topologies considérées. D'ailleurs, nous observons que l'optimalité dégrade lentement en fonction de β, et aussi avec une pente similaire (à peu près linéaire) dans tous les cas. Ça c'est un comportement souhaitable: comme il y a aucune transition de phase, ni aucun genou dans l'allure des courbes, on peut régler β entre une faible surcharge ($\beta \approx 0.3$) ou une haute qualité des chemins ($\beta \approx 0.7$).

Enfin, nous disséquons la raison derrière la sous optimalité des certaines chemins secondaires. On rappel que un chemin secondaire $\mathcal{S}_{i,j}$ est optimal s'il est le chemin le plus court et le plus diffèrent (en termes de chevauchement $\mathcal{P}_{i,j} \cap \mathcal{S}_{i,j}$) par rapport au première $\mathcal{P}_{i,j}$. Par conséquent, la sous optimalité du chemin secondaire peut être due (i)soit à un *chevauchement* entre les chemins primaires et secondaires $|\mathcal{P}_{i,j} \cap \mathcal{S}_{i,j}| > 0$, ou (ii) à un chemin plus long que le chemin secondaire optimal $\frac{length(\mathcal{S}_{i,j})}{L_{S'_{i,j}}} > 1$. Fig. 3.3 représente le chevauchement, c'est à dire le nombre de nœuds que les chemins primaires et secondaires ont en commun, conditionné par les chemins sous optimaux (les chevauchements des chemins secondaires optimales ne sont pas prises en compte). Comme montre la figure, la sous optimalité semble être liée à des nœuds en commun tel que $|\mathcal{P}_{i,j} \cap \mathcal{S}_{i,j}| \in [1, 1.5]$. D'ailleurs, puisque le chevauchement moyenne est toujours tel que $|\mathcal{P}_{i,j} \cap \mathcal{S}_{i,j}| \geq 1$ pour tout β, nous pouvons conclure que les chemins qui se chevauchent sont nettement plus fréquents que les chemins plus longs.

Analyse du système dynamique

En cette partie, nous analysons les propriétés temporelles du système, l'examen de la durée du processus de découverte de chemin, et ses propriétés de terminaison. Comme le temps de propagation est la composante dominante du retard, nous nous attendons que les propriétés liées au temps soient affectées par l'extension géographique du réseau, avec une large variation de la performance sur les différentes scénarios de Tab. 2. Par conséquent, nous avons également développé un modèle pour recueillir des nouvelles intuitions sur les propretés liées a la taille du réseau, sans être par contre lié à des cas topologiques spécifiques.

Rapidité Pour évaluer la rapidité du calcul des chemins nous fixons $\beta = 0.7$ comme un bon compromis entre l'optimalité et le coût supplémentaire. Puisque le chemin primaire est rapidement établis dans Apl , le calcul des trajectoires converge quand un nœud ne met plus à jour son chemin secondaire (par rapport à la pseudo code de l'algorithme, ce correspond à la dernière exécution de la ligne 13). Comme fait précédemment, pour recueillir des performances objectives, nous laissons chaque nœud démarrer une annonce, en prenant la moyenne sur 20 simulations. Lors de

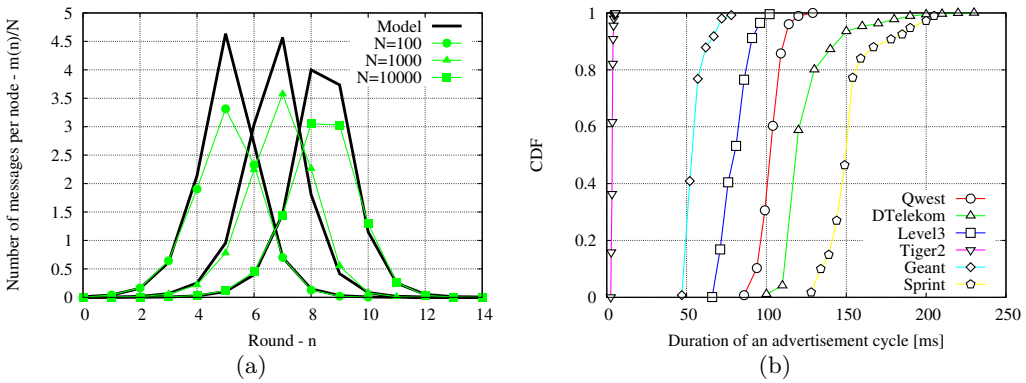


Figure 5: Dynamique des messages pendant un processus d'avertissement(a). Durée d'un rond d'inondation(b).

chaque tour, on mesure le temps écoulé entre le début de le processus d'avertissement (à la source) et la convergence du parcours secondaire (à tous les nœuds). La fonction de distribution cumulative (CDF) des temps de convergence du chemin est indiquée dans Fig. 4.4 pour tous les réseaux. Comme prévu, la durée brute est principalement affectée par le retard moyen de lien. Pour tous les réseaux, sauf DTelekom (dont le retard de liaison moyenne $\delta = 17.2$ ms est beaucoup plus grand que celui d'autres topologies, cfr. Tab. 2), 90% des chemins secondaires convergent en moins de 60 ms, et en tout cas la convergence prend moins de 100 ms *dans le pire des cas* (alors que dans [39], la convergence des algorithmes à état du lien prend bien plus de 100 ms dans le *meilleur des cas*). Cela confirme la solidité de Aplasia, qui permet une convergence rapide aux trajets multiples optimales.

Terminaison automatique Nous étudions maintenant l'évolution temporelle de la propagation du message au cours d'un seul cycle d' Apl . Nous analysons la durée du processus d'annonce sur la réseau, avant que le *backoff* exponentiel éteigne le processus d'inondation. L'objectif n'est pas de prédire précisément le nombre total de messages envoyés, qui est de toute façon prédit par Eq. (2), mais pour estimer certaines propriétés temporelles critiques, comme par exemple l'instant où le processus d'inondation atteint un pic, le temps à lequel il disparaît, l'impact de la taille du réseau, etc. Nous considérons un seul annonceur et supposons des retards de propagation homogènes afin que le temps puisse être considéré échantillonné. Pour tous $n \geq 0$, le nombre moyen de messages $m(n)$ envoyés par les nœuds N dans le réseau au temps n vérifie l'équation récursive approximative:

$$m(n+1) \approx m(n)\delta e^{-(1-\beta)\bar{M}(n)} \tag{3}$$

L'approximation dans Eq. (3) est valable pour des grands graphes, disons $N \geq 100$. La qualité de l'approximation est illustrée dans Fig. 5(a), comparant des solutions numériques du Eq. (3) avec simulations de réseaux aléatoires, en variant la taille du réseau jusqu'à $N = 10000$, le *backoff* fixé à $\beta = 0,7$ et le degré $\delta = 4$ (matchs similaires sont obtenu pour d'autres paramètres). On normalise le nombre des messages sur la taille du réseau $m(n)/N$, afin de faciliter la comparaison des tailles de réseau hétérogènes. Le modèle est très proche à la forme de la propagation des messages réels, qui confirme que l'hypothèse de Poisson tient bien dans la pratique. Plus en détail, comme montre la Fig. 4.5(a), la dynamique des messages reflète une augmentation exponentielle initiale en raison de l'inondation (comme les compteurs sont initialement 0, donc certainement transmis depuis $\beta^0 = 1$). Dès que les mêmes trames sont reçues sur le réseau, le *backoff* exponentiel entre en action en ralentissant la croissance des messages, jusqu'à ce qu'un pic soit atteint (au rond $n_{peak} = \text{argmin}_n m(n) \geq m(n+1) > 0$), après quoi le nombre de messages décroît progressivement (et s'arrête complètement à $2n_{peak}$ environ). De l'image, on comprend que n_{peak} augmente (i) de façon logarithmique avec la taille du réseau N, ou (ii) de façon linéaire avec le diamètre du graphe. C'est intuitif, depuis que l'inondation ralentit lorsqu'un nœud commence à recevoir de multiples copies du message, ce qui est toujours le cas lorsque la longueur du chemin atteint le diamètre du réseau.

Implémentation *Click*

Nous avons mis en place les principales fonctionnalités du plan de données d'Aplasia en un routeur modulaire *Click* [40]. Nos modules mettent pleinement en œuvre le plan de traitement de trames de données et des fonctions de maintenance, mais ils partialement implémentent les fonctionnalités du plan de contrôle. Pour donner une idée de la complexité de la mise en œuvre, le modules *Click* d'Aplasia compte pour environ 5000 lignes de code, 24 classes et 65 fonctions.

Principalement, nous avons utilisé la mise en œuvre de *Click* pour la vérification fonctionnelle des principes de Aplasia. Le banc d'essai est composé de 7 ordinateurs équipés d'un processeur Intel Xeon E3110 dual-core qui tournent à 3.00GHz, équipé de 4 cartes Ethernet. Les PCs sont disposés dans une topologie bus, et sont reliés entre eux par deux 100Mbps Ethernet liens point-à-point. Dans le banc d'essai, nous utilisons seulement des cartes Ethernet comme émetteurs/récepteurs point-à-point entre toutes les couples de routeurs: autrement dit, aucune commutation ou d'autres fonctionnalités Ethernet sont utilisés. Dans notre configuration, le nœud d'origine fonctionne dans l'espace utilisateur (de sorte qu'il est facile d'accéder à des fonctions d'horodatage sans modifier le code de *Click*), tandis que tous les autres nœuds exécutent en mode *kernel*.

Nous concevons deux scénarios de tests simples et instructifs pour comparer la mise en œuvre de *Click*, visant à rassembler la durée de (i)la fonction de transfert du plan de données t_{FW} et (ii) le traitement des trames du plan de contrôle au cours du processus

t

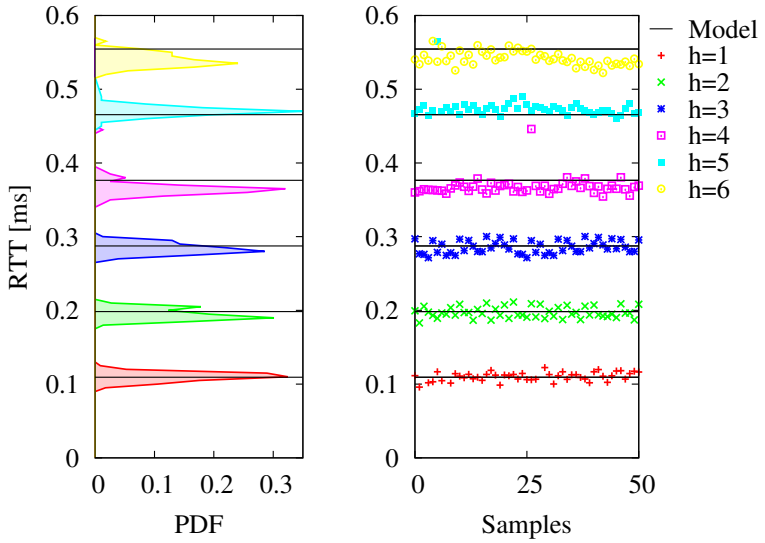


Figure 6: Fit du coût computationnelle du plane de donnée sous une implémentation Click.

de découverte t_{cp} . Nous déduisons le coût de traitement des messages dans le plan de données/contrôle en mesurant les temps aller-retour (RTT) à le différentes longueurs de chemin h.

Les résultats obtenus sont montrées dans Fig. 4.6: le temps de gestions et création des trames est $t_g = 54 \quad \mu$s et le temps de transmission est $t_{FW} = 45 \quad \mu$s, avec de très petites erreurs asymptotiques de 0,63% respectivement et 0,33 %.

D'ailleurs, nous avons fait des expériences avec l'annonce des chemins (dans une configuration légèrement modifiée), afin de recueillir le temps pour gérer et créer les trames du plan de contrôle t_{CP}. Les frais introduites par le plan de contrôle est de $t_{CP} = 48\,\mu$s (avec une erreur asymptotique de 0,2 %), et donc il est du même ordre de grandeur de l'opération d'acheminement dans le plan des données. Ces résultats soulignent l'intérêt pratique de Aplasia et la légèreté du processus d'annonce dans la tâche de contrôle.

Content Centric Networking

Entre les nombreuses propositions ICN, l'approche *Content Centric Networking* (CCN) [41] a suscité un intérêt considérable de la communauté scientifique. En tant que tel, et puisque un cadre unificateur conceptuel pour ICN est toujours en cours de définition au ICN Research Group (ICNRG), dans ce qui suit, nous plaçons notre travail dans le cadre

du CCN, en adoptant sa terminologie. En particulier, on décrive, analyse et modèle des algorithmes de mémoires caches sur des réseaux CCN.

Bien que les détails de CCN sont hors de la portée de cette introduction, nous décrivons en gros détails le système que nous utilisons. Un réseau CCN peut être considéré comme un réseau des caches dirigé par les clients. Dans ce qui suit, nous utilisons indifféremment les termes *requête* et *intérêt*, pour indiquer le message d'un utilisateur envoyé afin de récupérer un contenu donné. Chaque nœud a trois structures des données utilisées dans la phase d'acheminement: le *Pending Interst Table* (PIT) mémorise les interfaces entrant correspondantes aux intérêts acheminés. Le *Forwarding Interest Base* (FIB) est utilisé pour acheminer l'intérêt dans le réseau. Quand un client envoie un intérêt pour un fichier, chaque nœud du réseau décide le chemin que l'intérêt va faire en mémorisant le chemin qu'a déjà fait. La dernière structure implémentée dans un nœud CCN est le *Content Store* (CS): quand les donnés revient vers le client (en suivant les indications lassées dans le PITs de chaque nœud) chaque CS peut mémoriser les données traitée, ainsi que la successive fois qu'un client demande le même fichier il sera reprise dans le CS (qui assume le valeur de mémoire cache) sur le même parcours.

Nous considérons un graphe $G = (V, E)$ de $|V| = n$ caches de taille $C(v)$ $v \in V$, avec des utilisateurs qui demandent des contenus à un taux de $\lambda(v)$. Souvent, nous allons envisager des scénarios homogènes, indiquant avec C la taille du cache de chaque routeur et λ, le taux d'arrivée de chaque client, c'est à dire $C = C(v)$, $\lambda = \lambda(v) \quad \forall v \in V$.

Le catalogue $\mathcal{N}$ représente l'ensemble de tous les objets possibles i qu'un utilisateur peut demander. La taille du catalogue est indiquée par $N = |\mathcal{N}|$. En considérant un fichier $i \in \mathcal{N}$, soit $p(i)$ la popularité du contenu (c.-à-d., la probabilité qu'un contenu i soit demandé). Les objets peuvent être partagés en blocs. Dans ce dernier cas, nous définissons $d(i)$ la taille en blocs de contenu $i \in \mathcal{N}$, et nous avons $P(Di = k) = \frac{1}{D}\left(1 - \frac{1}{D}\right)^{k-1}$, c'est à dire , dans le cas des fichiers partagés en blocs, la taille de chaque objet a une distribution géométrique, avec une moyenne de D blocs.

Chaque contenu i est stocké en permanence dans un serveur (aussi dit gardien ou dépositaire ou dépôt). On note $\mathcal{S}(i)$ le dépositaire pour le contenu i .

Nous négligeons le *naming*, et les aspects de sécurité de CCN: chaque contenu i est représenté par son rang de popularité (c.-à.-d., le contenu 1 est le plus populaire, et ainsi de suite). La popularité du contenu i est distribué par une distribution Mandelbrot-Zipf: $p(i) = \frac{K}{(q+i)^{\alpha}}, \quad K = \left(\sum_{i=1}^{N} \frac{1}{(q+i)^{\alpha}}\right)^{-1}$. α est dit le facteur de forme de la distribution, car il indique la pente de la distribution à une échelle logarithmique. Lorsque $\alpha << 1$ le Zipf s'approche à une distribution uniforme. q représente le plateau de la distribution MZipf, et plus il augmente plus le nombre $p(i)$ tend vers une distribution uniforme. Dans Tab. 3, nous montrons l'espace des paramètres que nous étudions dans ce travail.

Finalement, nous définissons un *Network Caching Algorithm* (NCA) sur un réseau de mémoires caches comme un triplet $\langle \mathcal{F}, \mathcal{D}, \mathcal{R} \rangle$ de stratégies pour *l'acheminement*, le *meta-caching* et le *remplacement*, respectivement. La stratégie d'acheminement $\mathcal{F}$ est utilisée

Paramètre		Explication	Valeurs(octets)
Network		Taille de chunk	10 Ko
	C	Taille de Cache	10^6 chunks (10 Go)
	n	Taille du réseau	
		Topologie	Générique (réelle),Arbre , Grille, Torus
Catalogue	N	Taille du catalogue	10^8 files(10^{15} octets)
	D	Taille de fichier	10^3 chunks (10 Mo, geom.)
	$\frac{C}{ND}$	Cache/catalogue ratio	$[10^{-5}, 10^{-1}]$
Popularité	α	Facteur de forme	{1,1.5}
	q	Mandelbrot-Zipf plateau	{0,5}
Caching	$\mathcal{F}$	Forwarding	SPR, CATT [12], NDN [42], iNRR§7.3,NRR§7.3, Multipath§7.2
	$\mathcal{D}$	Meta-caching	LCE,FIX [15],LCD [16],ProbCache [17], BTW [19]
	$\mathcal{R}$	Replacement	FIFO,LRU,RND,BIAS

Table 3: Notation et valeurs de défault.

par chaque nœud pour établir le chemin dont envoyer les requêtes. La stratégie de *meta-caching* est utilisée par chaque nœud pour décider si stocker effectivement les données qui retournent vers le client. La stratégie de remplacement est utilisée pour établir, lorsque une décision positive a été prise, l'objet à effacer dans la mémoire cache (dans le cas que la mémoire cache est pleine).

Dans Fig. 6.2(b), nous inspectons l'impact de la mise en ouvre des politiques de cache considérant $(\alpha, q) = (1.5, 0)$. Nous rappelons que le choix d'un tel distribution de popularité n'est pas destiné à une évaluation absolue des performances d'un NCA, mais plutôt à une évaluation de l'impact relative de plusieurs paramètres, et elle est donc parfaitement justifiée.

Algorithmes conscients de la topologies

Nous évaluons maintenant les performances d'un NCA en prenant en compte différents scénarios topologiques, en déterminant ainsi à quel degré la topologie du réseau affecte les performances d'un NCA. En raison de la contrainte sur la vitesse d'un router CCN, il pourrait être donc utile d'examiner d'autres combinaisons outre la plus couramment utilisée ⟨LCE,LRU⟩. Par conséquent, nous évaluons l'impact de différentes topologies en considérant une stratégie aléatoire ⟨FIX($\frac{9}{10}$),RND⟩. Fig. compare le *cache hit* sur (i) un arbre binaire, (ii) les 6 topologies de Tab. 5.1, et (iii) montre la moyenne sur toutes les topologies. On peut constater que, en moyenne les performances entre ⟨LCE,LRU⟩ et ⟨FIX($\frac{9}{10}$),RND⟩ sont difficilement distinguables: aussi, ⟨LCE,LRU⟩ *n'est pas* toujours la meilleur choix sur toutes les topologies. Nous observons que, comme récemment montré

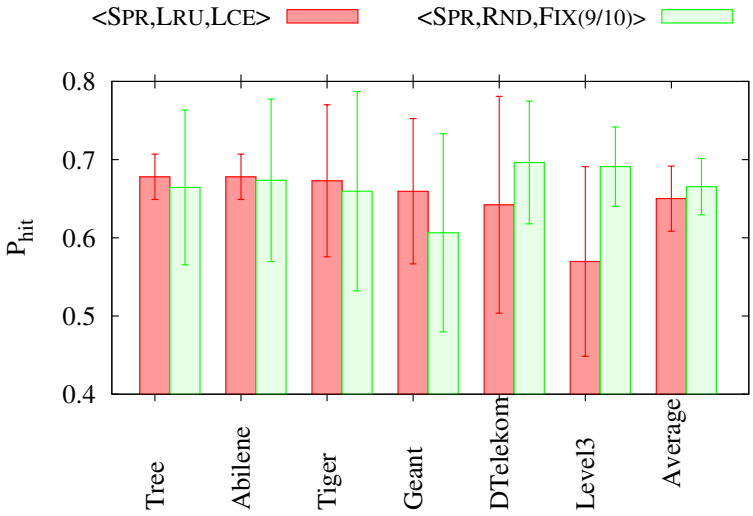


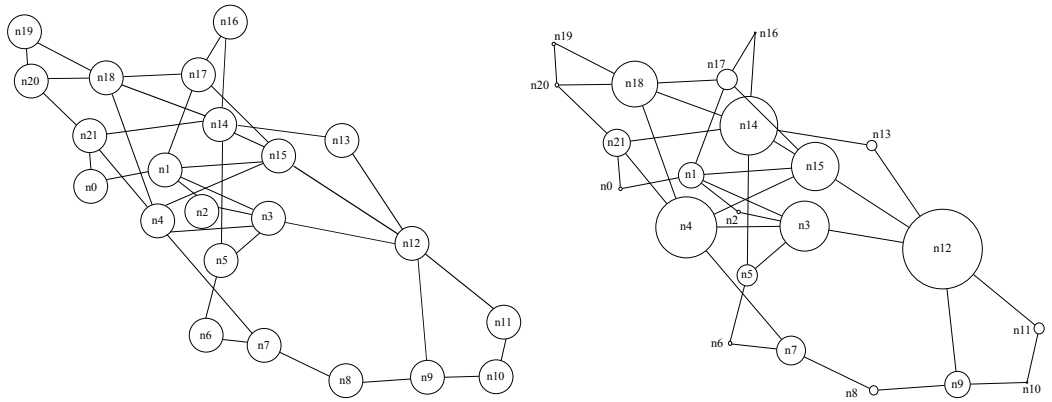
Figure 7: Comparaison de l'efficience d'un NCA sur différentes topologies.

dans [2], en utilisant un triplet ⟨SPR,LCE,LRU⟩, la topologie a une influence mineure sur les performances d'un NCA. Dans cette section, nous remarquons les conclusions de [2], mais on essaye d'exploiter les propriétés topologiques comme un moyen pour augmenter les performances des NCA.

L'information topologique peut être exploitée à plusieurs couches, et par multiples plans. Dans le rappel de cette paragraphe, nous illustrons le cas d'un planning de réseau CCN dépendant de la topologie. Plus précisément, nous considérons plusieurs mesures de centralité dans un graphe et allouons l'espace de mémoire cache dans une façon *hétérogène* sur le réseau CCN et on compare avec les performances d'une allocation *homogène*.

En étendant le travail de [43] nous calculons plusieurs mesures de centralité dans le graphe topologique G. Nous utilisons alors les valeurs de centralité de chaque nœud comme une base pour des stratégies différentes pour distribuer une quantité hétérogène de mémoire cache, en mesurant le gain de performance (ou la perte) *relatif* à un réseau CCN homogène ayant le même montant global de cache. Pour chaque nœud de la topologie ci-dessus, nous calculons des différentes mesures de centralité, à savoir le *Betweenness Centraility* (BC), *Degree Centrality* (DC) , le *Stress Centrality* (SC) , le *Closeness Centrality* (CC), le *Graph Centrality*(GC) et le *Excentricity Centrality*(EC) .

Nous définissons C_{tot} comme la taille globale des mémoires cache dans la topologie. Dans le cas d'un réseau homogène, nous fixons la taille des mémoires cache individuels $C(i) = 10\,\text{Go}$ [44]. Donc, dans le cas des réseaux homogènes, $C_{tot} = nC(i)$ avec $|V| = n$ le nombre de nœuds dans le réseau. Dans le cas des réseaux hétérogènes, nous



(a) Homogeneous (b) Heterogeneous

Figure 8: La représentation illustrée du dimensionnement des cache, avec la taille des cache CCN proportionnelle à la taille de nœud dans le graphe correspondant. Les images montrent un dimensionnement (a) homogène et (b) hétérogènes (dans le dernier cas, proportionnel à la *Betweenness Centrality* (BC).

exploitons les métriques de centralité comme suit. Considérons une métrique générique $X \in \{CC, GC, DC, EC, SC, BC\}$, où on note $X(i)$ la valeur de X pour le nœud $i \in V$ selon la topologie considérée.

Nous adoptons donc deux critères de dimensionnement des caches:

$$C_X^P(i) = C_{tot} \frac{X(i)}{\sum_{j \in V} X(j)}, \forall i \tag{4}$$

$$C_X^Q(i) = \max\left(c, \lceil C_X^P(i)/c \rceil c\right) \tag{5}$$

On remarque que l'Eq. (4) correspond à un parfait critère proportionnelle, lorsque la taille de mémoire cache $C_x^p(i)$ est distribuée au nœud i-ème proportionnellement à la métrique $X(i)$ normalisée sur la somme des mesures $X(i)$ sur l'ensemble des nœuds.

Alors l'Eq. (4) est une stratégie idéale qui permet de mesurer l'importance relative des scores de centralité, nous reconnaissons qu'il peut être difficilement réalisable dans la pratique: en effet, les modules de mémoire CCN seront quantifiés en multiples d'un module unitaire c, comme c'est le cas pour la mémoire RAM de nos jours. En tant que tel, nous considérons également une stratégie d'échantillonnage Eq. (5), où la taille des caches individuels $C_X^Q(i)$ est multiple de $c = 1\,\text{Go}$. Dans Eq. (5), on assume un modèle où les FAI vont investir dans un nombre fixe de modules de mémoire C_{tot}/c qu'ils peuvent ensuite déployé arbitrairement dans le réseau. Le point de vue que nous adoptons est qu'un FAI peut vouloir réaffecter le C_{tot}/c modules à sa disposition (par exemple, passer d'une

configuration homogène à une hétérogène), afin d'optimiser les performances réalisables sans encourir dans des coûts économiques.

Fig. 8 représente des nœuds de taille variable, dont le rayon est proportionnel à la taille de mémoire cache $C(i)$, pour visualiser où dans le réseau de la ressource de cache ont été alloués. Plus précisément, Fig. 8 contraste un réseau Géant homogène où $C(i) = C_{tot}/|V|$ pour tous les nœuds (graphe à gauche), contre une allocation hétérogène $C^P_{BC}(i)$ (graphe à droite).

On peut observer que le processus d'échantillonnage induit une erreur telle que le montant global de la mémoire cache est maintenant $C_{tot}(1+\epsilon)$ avec $\epsilon \in \mathbb{R}$ l'erreur induite par le processus de quantification. Cette erreur est due à deux opérations contrastantes: l'opération de max imposant une taille minimale du cache $C^q_x(i) \geq c \forall X, i$ incrémente ϵ, tandis que l'opération de plafond réduit ϵ. Finalement, l'erreur moyenne est de $E[\epsilon] = 2.3\%$. C'est possible que la différence en termes de performances (par exemple, le gain de *cache hit*) peut être due à un écart dans la taille du cache estimée (par exemple, quand $\epsilon > 0$). Pour écarter cette possibilité, nous vérifions l'absence de corrélation entre ces anomalies. Plus formellement, notons H_{Const} la probabilité de *cache hit* sur un réseau homogène, avec la taille globale de mémoire cache C_{tot} . Notons ensuite par $H^Q_X = (1+\gamma)H_{Const}$ le *cache hit* d'un réseau hétérogène avec des stratégie d'allocation de mémoire cache échantillonné selon la mesure de la centralité X, avec la taille globale du cache $\sum_{i \in V} C^Q_X(i) = (1+\epsilon)C_{tot}$. Le coefficient de corrélation $\rho_{\gamma,\epsilon}$ entre la série de paires (γ, ϵ) collectées sur tous les réseaux et les mesures de centralité, équivaut à $\rho_{\gamma,\epsilon} = 0,05$, en excluant des corrélations entre ces erreurs. En tant que tel, les différences de performance dans les domaines suivants dépendent uniquement des mesures de centralité.

Le but de notre campagne de simulation est (i) d'évaluer si un dimensionnement hétérogène de mémoire cache peut fournir des avantages de performance par rapport à un réseau homogène et (ii) si le gain de performance est cohérente dans toutes les topologies pour une répartition métrique $X \in \{CC, GC, DC, EC, SC, BC\}$.

Nous observons que l'échantillonnage joue en fait un rôle bénéfique. Prenons le métrique de *cache hit* H et notons H^P_X (H^Q_X) le *cache hit* atteint pour une topologie et une lois de popularité donnée, en utilisant une répartition proportionnelle (échantillonné) en fonction d'une métrique de centralité X. On définit alors l'erreur relative induite par l'échantillonnage sur le *cache hit* comme $(H^Q_X - H^P_X)/H^P_X$, qui est représentée dans Fig. 9(b) pour toutes les topologies et tous les métriques. Sur le graphe, une valeur négatif (zone grisée) correspond au cas (combinaisons des métrique et topologie) où la répartition proportionnelle donnerait de meilleurs résultats. Fig. 9(b) montre que la répartition proportionnelle donne de meilleures performances dans certains cas (partie gauche), mais la différence de performance avec l'allocation échantillonné correspondant est minime (moins de 2%). Inversement, il y a des cas dans lesquels la quantification peut apporter un gain presque de 20 % par rapport à une répartition proportionnelle. Essentiellement, cela est dû au fait que certaines mesures peuvent allouer une très faible quantité d'espace de mémoire cache pour certains nœuds, qui sont "corrigées" en ayant un minimum de cache dans le processus de

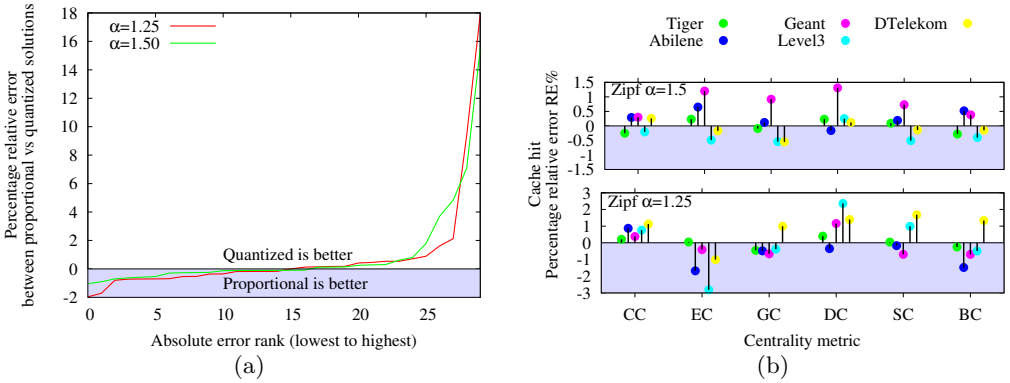


Figure 9: (a) Pourcentage d'erreur relative $(H_X^Q - H_X^P)/H_X^P$ sur le *cache hit* dans le cas d'une allocation proportionnelle et quantifiée. (b) *Cache hit* avec une stratégie d'allocation quantifié, pour diffèrent topologies et facteurs de forme. L'axe Y reports le pourcentage d'erreur relative$(H_X^Q - H_{Const})/H_{Const}$ par respect à une allocation homogène. La zone colorée corresponds à des pertes de performance dans le cas d'une allocation hétérogène.

quantification. Dans ce qui suit, nous considérons donc que les allocations échantillonné: ce choix est à la fois solide (car nous évitons des valeurs trop petits en raison de biais dans les mesures de centralité) et réaliste (comme une allocation parfaitement proportionnelle n'est pas directement applicable).

Nous soulignons que les paramètres de α peuvent en outre modifier les résultats sur une topologie donnée. Nous illustrons la situation en tenant compte de l'erreur relative entre le *cache hit* avec une stratégie d'allocation quantifié induite par la métrique X contre une répartition homogène constant, c'est à dire $(H_X^Q - H_{Const})/H_{Const}$ selon notre terminologie précédente. L'erreur relative est représentée dans Fig. 9(a) pour $\alpha = 1.5$ (en haut) et $\alpha = 1.25$ (en bas) pour toutes les topologies (points) et les mesures de centralité (en abscisse).

On observe que, pour les deux $\alpha = 1.5$ et $\alpha = 1.25$, les mesures DC améliorent le *cache hit*, sauf dans le cas d'Abilene, où la perte de performance est cependant très limitée (les performances sont très proches à la ligne de base de répartition constante). Cela peut s'expliquer par le fait que, dans la topologie Abilene, le degré ne varie pas significativement entre les nœuds, de sorte qu'une allocation proportionnel pratiquement dégénère en une répartition (presque) homogène. En revanche, dans les autres cas $(H_{DC}^Q - H_{Const})/H_{Const} > 0$ de sorte que certains gains peuvent être recueillies à partir d'une répartition hétérogène.

Enfin nous remarquons que le gain de performance est limité par un modeste 2,5% dans le meilleur des cas (topologie Level3 avec $\alpha = 1,25$ dans l'image en bas de Fig. 9(a)). En raison de ce gain limité sur les tailles de cache homogènes, il semble tout à fait qu'il n'y

ait pas de réelle incitation à utiliser des stratégies d'allocation hétérogènes.

Acheminement

Nous considérons maintenant la stratégie d'acheminement $\mathcal{F}$, composant de la triple $\langle \mathcal{F}, \mathcal{D}, \mathcal{R} \rangle$ d'un NCA. L'espace de design que nous considérons est résumée dans Tab. 4. La stratégie d'acheminement de requêtes $\mathcal{F}$ est limitée par les informations disponibles dans le tableau de routage(FIB).

Dans les cas que le nœuds CCN ont à leur disposition des informations utiles pour transmettre les requêtes vers un (ou plusieurs) serveur où une copie persistante de stockage des données (par exemple, le chemin le plus court vers un serveur donné), ces informations peuvent être exploitées par l'architecture CCN. Le contenu peut être trouvée "en route" (c.-à-d., le parcours des caches entre le demandeur de contenu et le serveur du contenu), en manquant les copies dans les mémoires caches voisines (par exemple, qui se trouvent le long du chemin le plus court d'un autre demandeur). De même, les données seront mises en cache seulement sur le chemin entre le gardian du contenu et le demandeur (ou client).

À l'autre extrême , dans le cas où aucune information utile est disponible en FIB, le voisinage doit être exploré pour trouver une copie temporaire. Dans ce cas, les demandes sont exprimées sur éventuellement plusieurs chemins. Ça peut conduire à les inconvénients habituels des algorithmes d'inondations (et donc exigeant les contre-mesures habituelles, comme les approches basés sur le TTL ou l'élagage probabiliste de certaines branches dans le processus d'exploration). Par contre, les copies temporaires seront alors disponibles à de multiples nœuds voisins.

Une autre dimension qui peut différentier la mise en ouvre d'une stratégie d'acheminement dans une réseau des mémoires caches est l'*unité de données minimal*. Aux deux extrêmes antipodes, le contenu d'intérêt peut être soit monolithique (mode objet) ou partitionné en morceaux ou *chunks* (mode morceau). De toute évidence, il y a une surcharge déterministe quand CCN est utilisé en mode morceau, puisque des demandes multiples doivent être exprimées pour récupérer les mêmes données. Au même temps, les demandes sont généralement de taille limitée, et le mode morceau offre une plus grande flexibilité pour récupérer les données. D'ailleurs, les demandes de la première partie de n'importe quel objet pourraient être exprimées selon un paradigme d'exploration: cela conduirait à la découverte du chemin vers une copie de l'objet le plus proche (en cache), qui pourrait être exploité par des demandes de morceaux successifs du même objet.

Stratégies d'exploitation

Afin d'étudier les stratégies d'exploitation, dans cette section, nous supposons qu'un algorithme de routage externe (et un système de résolution de noms) fournit à la couche d'acheminement des parcours alternatifs de demandeur à l'origine (le serveur) du contenu. Nous ne considérons que 2 chemins alternatives. La couche d'acheminement doit ensuite

FIB Knowledge	• None • Omniscient • Partial
Request forwarding strategy	• Exploration • Exploitation • Hybrid
Data unit	• Monolithic object-level • Partitioned chunk-level

Table 4: Espace de design CCN.

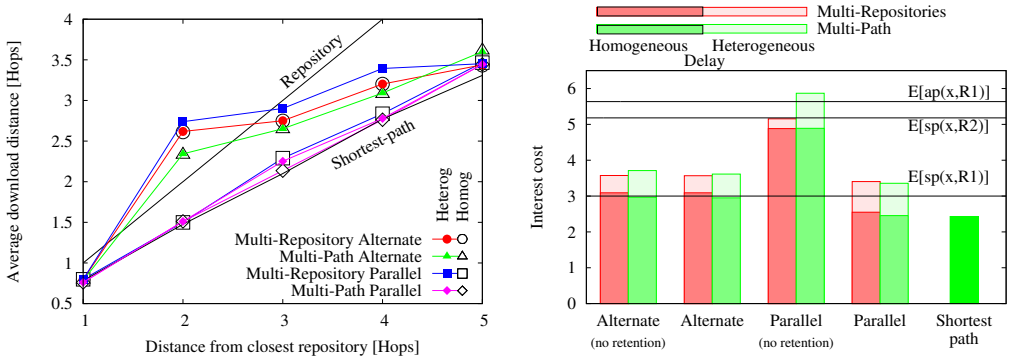


Figure 10: On montre l'impact du strategy layer sur la distance moyenne et les frais d'intérêts/données.

décider comment utiliser ces chemins, en prenant décisions morceau par morceau.

Nous comparons plusieurs stratégies alternatives des chemins multiples par rapport à le cas de base d'un seul chemin (le plus court). Plus précisément, nous considérons qu'il y a (i) deux chemins alternatives vers le même serveur, ou (ii) différents chemins singles vers plusieurs serveurs.

Dans les deux cas, la couche stratégique doit prendre autres décisions concernant le processus de sélection de parcours, tels que (i) un nœud peut explorer les deux voies en parallèle, ou (ii) en alternance entre les chemins (et dans ce cas, si uniformément au hasard ou déterministe comme en round robin). Le compromis dans ce cas est entre (i) une charge d'intérêts qui viole le flux d'équilibre et peut entraîner des contentions dans les caches et (ii) une convergence peut-être plus lente vers le serveurs.

Fig. 10(a) indique la distance moyenne de téléchargement en fonction de la distance à partir du serveur le plus proche. Les points remplis se réfèrent à un scénario avec des retards hétérogènes. Les points vides à des retards homogène. Les deux lignes de référence

reportent les cas où le contenu est entièrement téléchargé depuis le serveur le plus proche, ou est téléchargé avec SPR, respectivement. Fig. 10(b) indique le charge d'intérêt moyen, c.-à-d., le nombre de liens que chaque intérêt a traversé avant de frapper un cache. Le couleur clair est utilisé en cas de retard hétérogène, foncé pour le retard homogène. Les trois lignes indiquent la distance moyenne vers le dépôt le plus proche ($E[sp(x, R1)]$), la distance la plus courte vers le serveur alternatif ($E[sp(x, R2)]$) , la longueur du chemin alternatif vers le dépôt le plus proche ($E[ap(x, R1)]$).

Comme prévu, lorsque plusieurs chemins sont utilisés, en étant plus longs de la distance minimale, les distances de téléchargement augmente. En regardant Fig. 10(a), cet effet est particulièrement visible pour les nœuds qui sont proches au dépôt, avec un pic à deux sauts. À l'inverse, les nœuds lointains trouveront probablement le contenu mis en mémoire cache quelque part avant le dépôt, avec des performances qui rapprochent le routage SPR.

Lorsque plusieurs chemins sont utilisés en parallèle, la charge d'intérêt est à peu près double. En considérant Fig. 10(b), si on essaie le chemin en parallèle juste pour le premier morceau, on peut atténuer efficacement l'augmentation de la charge d'intérêt, au prix de préserver le chemin par chaque objet dans les routeurs CCN .

Nous rappelons que dans le cas où des chemins alternatifs sont utilisés, il n'y a pas de différence entre un critère aléatoire uniforme et un déterministe round robin (donc, nous ne montrons pas cela dans les dessins). Intuitivement, si les décisions déterministes ne sont pas coordonnées entre les nœuds, elles sont équivalents aux décisions aléatoires dans la perspective globale du réseau. Au contraire, quand plusieurs chemins sont utilisés dans une façon parallèle, cela conduit à une politique plus agressive, qui a plus de chances de trouver le contenu mis en cache plus proche.

Nous constatons que plusieurs chemins alternatifs vers le même serveur doivent être préférés à l'utilisation de plusieurs chemins singles vers plusieurs serveurs (ça conduit à des choix plus robustes en cas de retard hétérogène). Cela malgré le chemin alternatif moyen vers le serveur plus proche est en moyenne plus long que le plus court chemin vers le serveur alternatif, c'est-à-dire, $E[ap(x, R1)] > E[sp(x, R2)]$ en Fig. 10(b).

Dans l'ensemble, nous constatons que les stratégies d'acheminement naïves qui exploitent différents chemins (autre au le plus court) peuvent conduire à explorer une distance plus longue -en augmentant la charge globale du réseau, et en réduisant la probabilité de *cache hit*. Au même temps, nous reconnaissons que les avantages de cette stratégie proviennent d'une meilleure résilience (dans le cas le plus court chemin sûr) et d'un charge éventuellement réduit au serveur d'origine (dont la bande passante peut être une ressource rare par rapport aux capacités du routeur). Donc, il semble intéressant explorer des stratégies hybrides qui complètent le routage avec une exploration opportuniste d'un voisinage de nœuds CCN. Nous ainsi évitons que la distance augmente, en laissant que la requête trouve (idéalement) la plus proche copie dans la réseau.

Stratégies d'exploration

Dans cette paragraphe, nous abandonnons les approches d'exploitation, pour nous concentrer sur des techniques d'exploration. Comme on a dit ci-dessus, les stratégies d'exploitation augmentent moyennement (par rapport au chemin le plus court) la distance parcourue par contenu. Ça c'est car les données sont forcés sur des chemins prédéterminés. Au lieu de cela, un protocole de routage idéal devrait considérer toutes les copies temporaires sur le réseau, pour transmettre les requêtes des utilisateurs vers la meilleure copie disponible (par exemple, le plus proche). Nous appelons cette stratégie idéale (*Ideal Nearest Replica Routing* (iNRR)

Ainsi, nous abordons le problème de la définition, l'analyse et la mise en œuvre d'une politique d'acheminement $\mathcal{F}$ d'exploration, adapte au CCN, qui se rapproche cependant vers l'acheminement iNRR. Notre objectif est de concevoir une stratégie de transport, que:(i) peut découvrir les copies du contenu temporaires, (ii) exige peu ou pas de connaissance a priori, (iii)ne génère pas trop de surcharge de signalisation supplémentaire; (iv) peut réaliser une coordination implicite de cache.

Nous considérons une grille 10x10 (100 nœuds) et un arbre binaire de 6 niveaux ($2^6 - 1 = 63$ nœuds). Contrairement à [2], puisque les réseaux sont conçus en respectant la tolérance aux pannes et les principes de la résilience, il est extrêmement improbable pour une topologie d'accès d'avoir exactement un seul lien entre n'importe quelle paire de nœuds parents et enfants - sinon, couper un seul lien dans la hiérarchie aurait coupé une sous arborescence. En tant que tel, nous considérons qu'un nœud peut avoir un lien supplémentaire avec son oncle (le frère de son père direct) qui peut être aussi utilisé pour la sauvegarde ou l'équilibrage de charge. Nous modélisons la présence de ces liens supplémentaires par une probabilités $\mu \in [0, 1]$.

Nous utilisons un rapport entre l'espace de mémoire cache et la grandeur du catalogue de 0,1% instancié dans un petit (grand) scénario où les mémoires caches sont capables de stocker 100 (100,000) objets à partir des catalogues de 100,000 (100,000,0000) objets. Ça nous permet d'explorer respectivement une grande range des réglages, et de recueillir les performances sur un cas d'utilisation plus réaliste.

Comme mesure de performance, nous ne considérons que la distance moyenne que le contenu a parcouru dans le réseau CCN. Cette mesure a l'avantage d'être très effective et compact en même temps: elle se rapporte à la qualité du service d'utilisateur (le retard) ainsi que la qualité du service du réseau (c.-à-d. charge et *cache hit*).

En termes de $\mathcal{F}$, au lieu d'être limité par la mise en œuvre (et la configuration) des précisions sur les nombreux NCA politiques proposées [9–13], nous ne considérons que (i)iNRR [2], comme la limite supérieure de le performances réalisables pour une stratégie d'exploration, et (ii) SPR que nous nous attendons à correspondre au limite inférieure pour les stratégies d'exploitation. En termes de *meta-caching* $\mathcal{D}$, nous mettons en œuvre plusieurs propositions contenues dans Tab. 3. Nous incluons LCE comme terme de comparaison, qui fourni une représentation de la limite inférieure des performances , car il

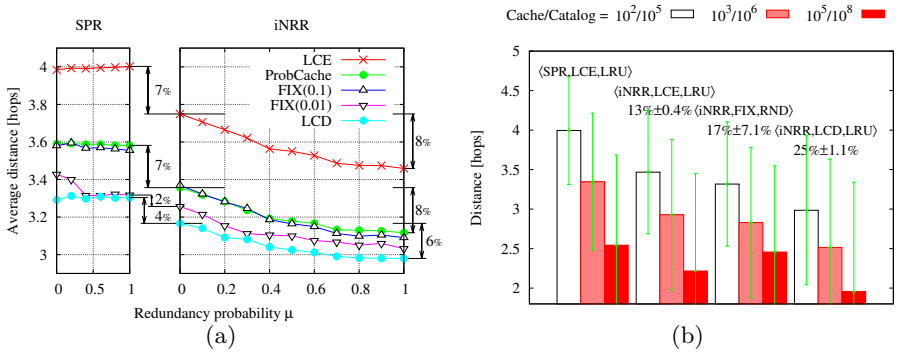


Figure 11: (a)Analyse de sensibilité $\langle \mathcal{F}, \mathcal{D} \rangle$ avec $\mathcal{R}$=LRU: arbre binaire avec 6 niveaux, avec une probabilité de redondance variable μ. (b) Scénarios réalistes.

fournit peu de diversité entre les mémoires caches, et force un taux d'effacement élevé sur tout le réseau. Enfin, en termes de remplacement $\mathcal{R}$ nous expérimentons soit avec LRU et soit avec un remplacement probabiliste uniforme [44, 45]. On remarque que dans cette paragraphe on suppose des objets entières (mode objet).

Performance Idéal Fig. 11(a) rapporte une analyse de sensibilité des politiques de méta-caching, recueillies par simulation en modulant légèrement la redondance du réseau $\mu \in [0, 1]$. Le graphe est annotée avec le gain obtenu en passant d'une stratégie $\langle$SPR,$\cdot\rangle$ à une stratégie $\langle$iNRR,$\cdot\rangle$, ainsi que avec le gain du à la redondance (à partir de $\mu = 0$ pour $\mu = 1$ pour chaque $\langle \mathcal{F}, \mathcal{D} \rangle$). Comme on peut s'y attendre, la redondance joue un rôle négligeable pour la stratégie SPR (bien que dans le cas de plusieurs chemins équivalents, en choisissant SPR entre eux au hasard, il peut traverser différents caches). Sans surprise, les décisions LCD déterministes toujours obtient les meilleures performances pour les arbres [15], en présentant une bonne interaction avec iNRR. Ensuite vient des simples décisions probabilistes (FIX($\frac{1}{100}$)), tandis que des stratégies probabilistes plus complexes entraînés soit sur la distance (ProbCache [17]) soit sur des propriétés topologiques (par exemple, Btw [19]) qui atteignent des gains intermédiaire.

Les scénarios petites, moyennes (ou grandes) nous permettent d'explorer respectivement une large carnet de réglages des paramètres, et de recueillir les performances sur un cas d'utilisation plus réaliste (ou extrême). Fixons $\alpha = 1$ et le rapport entre le cache et la taille du catalogue C/N à 0,1%, et lassons varier soit le cache C que la taille du catalogue N. Précisément, nous considérons un scénario de petite échelle avec $C/N = 10^2/10^5$, de moyenne échelle avec $C/N = 10^3/10^6$ et de grande échelle avec $C/N = 10^5/10^8$. Comme les applications vidéo sont prééminentes, et compte tenu de la taille moyenne des vidéos YouTube de 10Mo [46] la taille de la cache varie

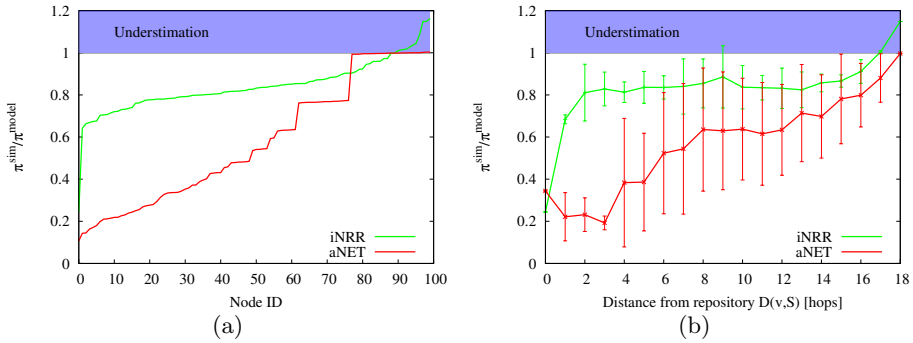


Figure 12: Probabilité de hit de iNRR et aNET exactitude par mode (en haut) et en fonction de la distance depuis le serveur(en bas). Topologie grille de 10x10 nœuds.

entre un scénario plutôt réaliste de 10Go [44,47] et un scénario plus futuriste de 10TB. La taille du catalogue du scénario moyen est du même ordre de grandeur de [2] , alors que le scénarios à grande échelle modèle un scénario YouTube plus difficile [48]. On peut observer dans Fig. 11(b) que les gains des scénarios de plus grande échelle sont plus grands que dans celui à petite échelle (ca c'est car la lois de popularité reste inchangée). Au même temps le gain relatif entre le différentes stratégies est toujours pareille pour les différents cas.

Le modèle Nous changeons l'ensemble des équations aNet [8] pour modeler la stratégie d'acheminement iNRR. Sous la stratégie SPR, le contenu peut être éventuellement trouvé seulement le long du chemin plus court vers un gardien du contenu. La différence cruciale avec aNet est que, avec la stratégie iNRR, chaque *chemin valide* est éventuellement suivi. Par chemin valide, on sous-entend que i) les chemins sont sans boucle , (ii) dans le cas où plusieurs copies sont stockées à plusieurs nœuds le long de tout chemin donné, la copie la plus proche est accessible. D'ailleurs, (iii) dans le cas de plusieurs copies ayant égale distance sur des chemins multiples, chaque exemplaire est également susceptible d'être choisi.

Quant à aNet, nous savons par [8] que l'impact de l'hypothèse IRM (Independent Reference Model) augmente avec la taille du réseau (ou, de façon équivalente, diminue avec la densité de serveurs sur le réseau). C'est parce que l'hypothèse IRM affects les longs trajets. Quant à iNRR, nous savons à partir des résultats précédentes qu'il raccourci considérablement la longueur de trajet moyen pour une copie en cache: en tant que tel, nous pouvons nous attendre un impact mineur de l'hypothèse IRM.

Nous considérons une grille 10x10, où le gain de iNRR sur SPR est visible. Par conséquent, nous nous attendons aNet d'être affectés négativement par la grande

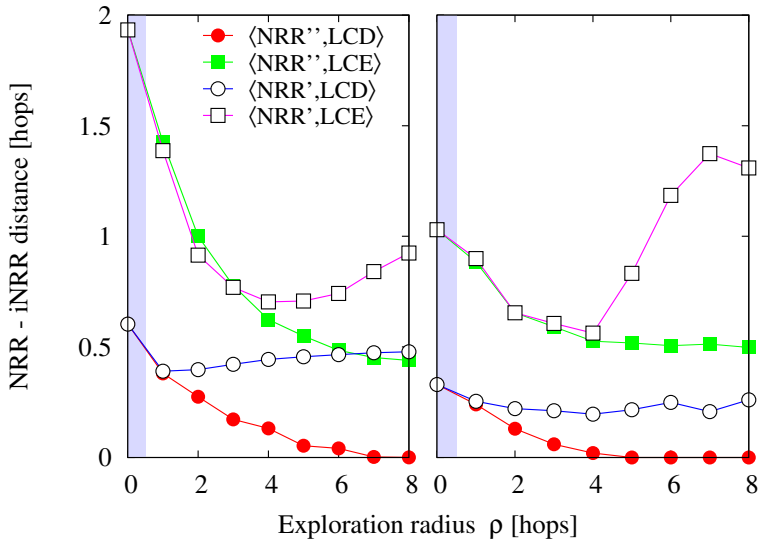


Figure 13: Distance additionnelle des implémentations NRR par rapport à iNRR. Acheminement NRR' et NRR", par LCE or LCD meta-caching, en fonction du radius d'explorationρ. Grille de 10x10 (à gauche) et arbre binaire de 6 niveaux complètement redondant $\mu = 1$ (a droit).

taille de la topologie, comme la distance au serveur pour SPR peut devenir assez longue. En revanche, puisque iNRR devrait trouver des copies plus proches, nous nous attendons un impact IRM violation (voir [8]) . Deuxièmement, sous iNRR les nœuds partagent leur flux dans chaque voisin: puisque on mélange des fluxes de défaut indépendants, l'approximation IRM a des affects mineurs par rapport au routage SPR (de façon similaire à ce qui se passe en augmentant le degré de l'arbre SPR dans [8]) .

Nous calculons la précision du modèle par rapport à la simulation pour (i) chaque nœud individuellement (Fig. 12(a)), ainsi que pour (ii) tous les nœuds ayant la même distance $\{x : D(x, \mathcal{S}) = d\}$ du serveur $\mathcal{S}$ (Fig. 12(b)). Plus précisément, en indiquant la probabilité de hit moyen pour le nœud v avec $\bar{\pi}_v$, nous évaluons la précision du rapport $\bar{\pi}_v^{sim}/\bar{\pi}_v^{model}$. Par souci de lisibilité, en Fig. 12(a) les nœuds sont classés pour rapports croissantes de $\bar{\pi}_v^{sim}/\bar{\pi}_v^{model}$. D'ailleurs, en Fig. 12, nous complétons le ratio moyen avec de barres d'écart type. Les résultats confirment que l'erreur iNRR est nettement inférieure de ce de aNet. Nous pouvons finalement observer, depuis Fig. 12, comme l'erreur iNRR est moins affectée, par rapport à aNet, de la position topologique de chaque nœud (essentiellement, la distance de SPR) .

Implémentation Il doit être clair que iNRR est une politique d'acheminement idéal, nécessitant un oracle ou, de façon équivalente, la connaissance de l'état de tous les caches. Nous proposons donc deux implémentations pratiquées que nous appelons Nearest Replica Routing (NRR). En particulier, nous nous concentrons sur la phase d' *exploration*, dont nous proposons deux implémentations alternatives basées sur des algorithmes d'inondations limitées, à savoir NRR' et NRR", qui nécessitent respectivement une et deux phases. Les deux NRR' et NRR" inondent le réseau avec des requêtes, en limitant la portée des inondations à travers un champ TTL qui on appelle le radius ρ.

Les différences entre NRR' et NRR" sont dans la façon dont les demandes sont traitées au cours de la phase d'exploration. L'algorithme NRR' *demande des intérêts réguliers*, afin qu'il génère peut-être plusieurs morceaux de données en retour - un pour chaque copie en cache trouvée à une distance mineur de ρ. Par conséquent, NRR' éventuellement génère un surcoût en termes de taux d'effacement en cache, bien que la durée de la phase d'exploration est la minimum possible avant que la copie la plus proche est touchée. Inversement, NRR" inonde la réseau de *méta requêtes*. Un drapeau est mis à indiquer que seule une réponse binaire concernant la disponibilité du contenu est demandé en retour.

Nous comparons NRR' et NRR" à iNRR en mesurant le nombre de sauts supplémentaires nécessaires en moyenne pour trouver le contenu. Pour être complet, on considère $\mathcal{D} \in \{$LCE,LCD$\}$ et $\mathcal{F} \in \{$ iNRR , NRR' , NRR"$\}$ et nous mesurons le nombre de sauts supplémentaires par rapport à la stratégie de CCN idéal $\langle$iNRR , LCD, LRU $\rangle$. Fig. 13 représente le nombre de sauts supplémentaires en fonction du rayon ρ pour la grille (à gauche) et l'arbre (à droite). Le dessin indique tous le combinaisons des paramètres: NRR" (coloré) vs NRR' (blanc), LCD (cercle) vs LCE (carré). On peut observer que pour $\rho = 0$, NRR dégénère en SPR routage (région en ombre). Plusieurs observations intéressantes sont recueillies à partir de Fig. 13. Tout d'abord, la performance de $\langle$iNRR , LCD, LRU $\rangle$ peut être arbitrairement proche de $\langle$NRR",LCD, LRU $\rangle$, comme la distance supplémentaire tend vers zéro pour ρ *ge*6, soit sur les arbres que sur les grilles. D'ailleurs, les effacements se traduisent par une importants perte des performances dans NRR'. Cela est dû à l'utilisation de paquets de requêtes régulières dans NRR', générant des données en retour qui comportent plusieurs effacements depuis le cache (même sous LCD). Enfin, on remarque que la distance supplémentaire diminue avec ρ croissant seulement dans le cas des NRR": cela signifie que l'exploration de NRR" est non seulement efficace, mais aussi robuste. Inversement, dans le cas NRR', quand ρ augmente, l'effacement augmente en raison de plusieurs chances de trouver le contenu sur des parcours plus longues.

Contents

Chapter 1

Introduction

Internet only just works [20]. In the current state, the Internet may be improved, either at "software layer" or at "hardware layer". In the former case, a set of IP-based protocols (e.g.,TCP, UDP) can be extended to exploit all the functionalities provided at lower layers . With respect to the latter case, IP can be deployed on almost every kind of physical layer: radio, fiber, copper, and henceforth. Each of these physical layers has different properties made available to higher layers by IP. In the following we disregard hardware solutions on the behalf of software ones.

In this model IP acts as the narrow waist [21, 22, 41] of a pictorial hourglass formed by different architectures/protocols for each layer (starting from the application, and ending in the physical layer). For instance, Fig. 1.1 reports the evolution of the higher part of the hourglass through the last 20 years. Indeed, while changing upper and lower layers is fairly simple [22, 49], changing IP (and then the network core) is an extremely challenging task (e.g., we are still updating the current infrastructure from IPv4 to IPv6 [50]).

Evidently, ISP operators are not disposed toward this task, unless research shows the real usefulness of such an approach. Nonetheless, to show the utility of an IP modification, there needs to be large scale deployment by a network operator. This vicious circle is hard to brake.

As a matter of fact, the actual usage of IP networks stratifies a series of scalability "patches" introduced when the system either ceases to work or presents serious and upcoming issues.

1.1 A bit of history

ARPANET represents the actual Internet's ancestor. It was developed in 1969 to interconnect a small researcher community [51], and was mainly based on the Network Control Program, NCP [52]. NCP provided host-to-host communication, and glued together addressing and data transfer (e.g., end-host communication, flow control and henceforth).

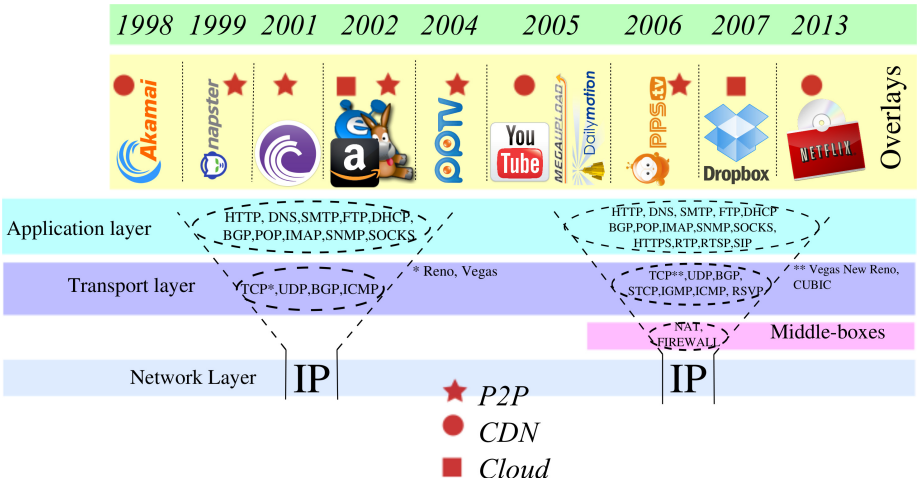


Figure 1.1: Pictorial evolution of the Internet over the last two decades.

Soon, it was clear that, for the sake of flexibility [53], machine addressing and file transfers should occupy two different layers. This conclusion had as consequence the birth, in March 1978, of two distinct protocols: the Transfer Control Protocol (TCP) and the Internet Protocol (IP) [53].

Since then, deploying IP over larger scale scenarios represents a continuous challenge for the Internet's architects, walking on the thin ice of infeasibility. Besides, given the actual Internet is composed by more than one billion machines [20], a strongly linked issue in modifying, updating, and thus changing the actual core protocols is to *incrementally deploy* those changes. Indeed, it would be actually impossible to switch off the whole set of machines (routers and end-hosts) to implement actual protocol modifications. Let's note that changes to higher layers (e.g., changes in TCP, HTTP, and henceforth) are definitely simpler: for instance, they can be introduced by gradually updating operative systems, leveraging on the layer transparency.

IP layer's rigidity has been widely analyzed in the recent research, and is known as the *Internet ossification* [20–22]. Moreover, the wide development of HTTP/TCP middle-boxes (e.g., NATs or firewalls) is leading to further ossify the stack around these protocols [22–24]. As a matter of fact, *deployable solutions for further scaling the Internet usually come as patches, implemented at higher levels of the TCP/IP communication stack.*

This situation has encouraged the development of *overlay applications*. Overlays provide a scalable and incrementally deployable solutions distributed *over* the top of the communication stack. An overlay network is a set of virtual application-level connections (i.e., not physical) built on the top of the lower networking layers. It's beyond the scope of this introduction to focus on the description of these applications. In Fig. 1.1 we report a series of well known overlay applications developed during these last 20 years of the In-

Year	Global Internet Traffic (GB/s)	Forwarding table size (Entries)
1992	0.001	25000
1997	0.02	50000
2002	100	100000
2007	2000	200000
2012	12000	500000
2017	35000	1200000

Table 1.1: Internet growth of the last two decades in terms of global traffic and routing table size.

ternet. Content Distribution Networks (CDNs), Peer to Peer (P2P), and, recently, Cloud Computing, represent the main overlay classes to which these applications belong. Each of the above applications happens to gain popularity for a certain amount of time, soon leaving the Internet scenario either for legal issues (we report the notorious Megaupload closure carried out by the FBI forces in 2009) or for lack of performance.

However, Internet ossification represents the root of most ills that are becoming more and more serious in today's scenario: security, management, performance, flexibility represent only a small subset of them. Even though the need to solve these problems still seems to lack urgency, outlining possible solutions appears mandatory for the networking community. Research could and should propose a more general framework for radically solving the scalability problems affecting the Internet for more than two decades. The only viable solution is to attempt a deployable modification of the network layer, borrowing ideas and solutions from the higher tiers, and porting them into the core.

In particular, in this Thesis we focus on two different scalability issues the Internet is facing:

- The routing tables growth, essentially due to the greater and greater number of hosts connected to the network.

- The increase of Internet traffic basically due to the wide distribution of content-based applications (e.g., YouTube, DailyMotion, and henceforth).

In order to have a numeric taste of the problems dimension, we show in Tab. 1.1 the global internet traffic (as reported in the Cisco VNI forecasts [26, 54]) and the size of the routers forwarding table (as reported in [25]) from 1992 to 2017. As we can see, in about twenty years we moved from few Megabytes to Exabytes ($1EB = 10^{18}\, bytes$) of global traffic and from tens of thousands to millions of routing entries.

1.2 Host vs Information Centric Networking

Actual research in the field of future Internet architectures is trying to clean the patchwork slate Internet has become. The high level goal is to borrow ideas and mechanisms from the overlay solutions, trying to bring these mechanisms within the Internet core. In particular, we focus on two of these approaches:

- Under the name of Host Centric Networking (HCN), we classify the umbrella architectures that focus on decoupling host identification and location. This opens the possibility of easily distributing the host name-space among the routers of the network alleviating the routing table explosion.
- Under the class of Information Centric Networking (ICN), we refer to the set of future Internet architectures that make content directly addressable within the networking layer, thus enabling caching as a means to cope with the traffic growth.

In the following we provide an overview of each approach, highlighting the points on which this Thesis is mostly focused. We detail the background of the HCN and ICN proposals in Ch. 2 and Ch. 5, respectively.

1.2.1 Host Centric Networking

Host Centric Networking (HCN) is a class of clean-slate approaches which aim at squeezing the routers forwarding tables by decoupling the location/identification pair that characterizes the IP protocol. The basic idea is to design an architecture based no longer on hierarchical names but rather on *flat labels*. Everything can be included in a flat label, and it is by design independent of the location of the objects. Some approaches even propose simple 48-bit Ethernet identifiers thus electing Ethernet as the future thin waist of the communication hourglass.

Thus, the base line of this approach is to identify a sort of virtual, global and authenticated name space on which to route host-to-host requests. Usually the basic strategy is to use a sort of Distributed Hash Table (DHT [55]) implemented within the networking layer and distributed between the border routers of the ISP. The DHT is usually queried to identify the successor of the host identifier in the DHT geometry. HCN approaches widely use *on-path caching* techniques, in order to store the best paths toward already visited hosts.

Host Centric Networking borrows additional theory from *compact routing* approaches. This latter is a branch belonging to graph theory which tries to scale routing on large topologies, while still minimizing the length of the determined paths. Compact routing, and hence most HCN proposals, usually neglect the low level intra-routing component (i.e., within ISP boundaries) usually left to a canonical OSPF-like protocol.

Indeed, our work in this field is mostly related to the *routing and forwarding* inside the ISP boundaries. We claim that flattening the network also impacts on the traditional

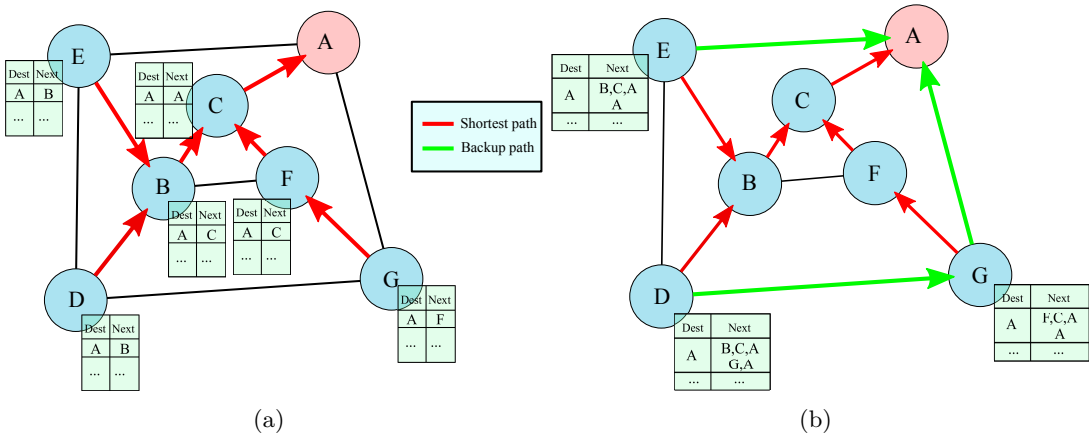


Figure 1.2: Pictorial comparison between the traditional hop-by-hop routing infrastructure (a) and Aplasia(b).

interior end-to-end (and not host-to-host) routing. Thus, we design, model and analyze the Adaptive Probabilistic Link-state Architecture for Switching In Autoforwarding (Aplasia). Aplasia is a forwarding architecture that leverages on flat names and is explicitly designed for the ISP interior. Aplasia is composed by two distinct parts:

- A path-finding algorithm based on the same flooding procedure adopted by OSPF. Nonetheless, trading off message with computational complexity, Apl is able to find more than one single path between two different end-points.

- An autoforwarding plane which *exploits* the paths found by Apl . Autoforwarding shifts FIBs toward the network edge, highly simplifying the core part of an ISP network.

In Fig. 1.2, we pictorially show the basic Aplasia's idea. Traditional intra domain forwarding (Fig. 1.2(a)) usually stores next-hop shortest path information in each node of the network. Aplasia(Fig. 1.2(b)) simplifies the network core, by shifting routing state toward the edge, and at the same time collecting more than a single path between each routers pair. The first part of this Thesis focuses on modeling and evaluating the two Aplasia's components describe above. In particular, we analytically model algorithmic costs, assess algorithmic performance by simulation, and experimentally evaluate autoforwarding benefits.

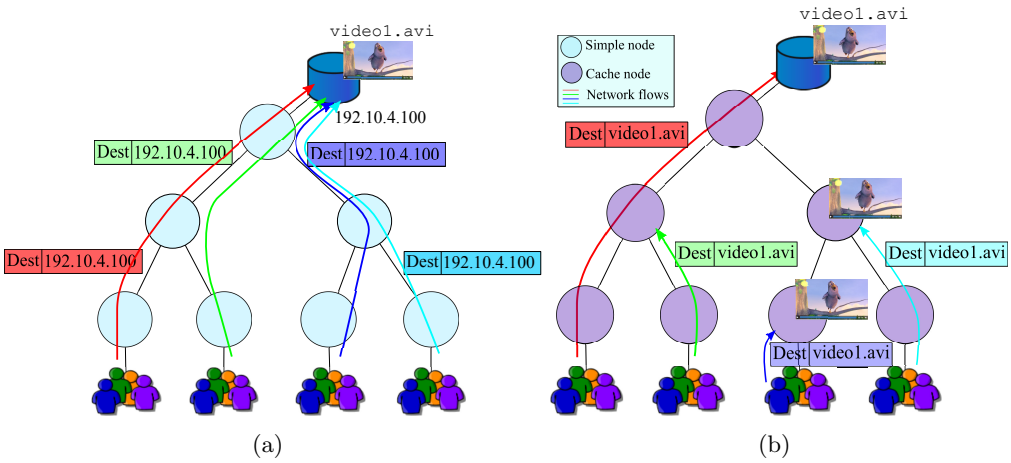


Figure 1.3: Pictorial comparison between the traditional host-based file retrieval (a) and the ICN paradigm (b).

1.2.2 Information Centric Networking

Nowadays, more than 60% of the Exabyte Internet traffic shown in Fig. 1.1 is consumed by Video applications [26], and it will exceed 90% in 2017. These data empirically show that the network is more and more becoming a means to download media content, rather than simply find hosts.

Most overlay applications, like P2P or CDNs, clearly try to cope with this content demand explosion. With the P2P approach, users find and download a file most of the time ignoring from which peer(s) the content has been downloaded. With the CDN approach, clients obtain contents (e.g., a YouTube video) from an arbitrary and unknown server.

Addressing this content-host indirection represents the core idea of Information Centric architectures. Indeed, ICN aims at replacing the actual thin waist of the hourglass with the concept of *content*, deploying a general purpose content-based protocol above the TCP/IP stack. Developing most of the current applications in a particular ICN context, would silently shift the current hourglass waist. This shift will make IP easily replaceable, in the same way that lower layers are easily replaceable for IP. In other words, the actual IP layer will be considered as a mere transport layer for a particular ICN protocol.

Most ICN approaches heavily rely on *caching*, often dealing with receiver oriented network of caches rather than a simpler network of routers. In Fig. 1.3 we pictorially show ICN principles. Traditional best effort approaches (Fig. 1.3(a)) let all the flows query the server hosting `video1.avi`. Instead, in information-oriented approaches (Fig. 1.3(b)) flows are filtered either by request aggregation (nodes on the left) or by caching (nodes on the right). Roughly speaking, since the name of the requested content is specified within

Chapter	Subject	S	M	E
Ch. 2	HCN background			
Ch. 3	APL analysis and description	⋆	⋆	
Ch. 4	APLASIA	⋆	⋆	⋆
Ch. 5	ICN background			
Ch. 6	Caching assessment	⋆		
Ch. 7	Coupling caching & forwarding	⋆	⋆	
Ch. 8	ccn$\mathcal{S}$im performance analysis	⋆		
Ch. 9	Conclusions & Future works			

Table 1.2: Thesis synopsis. For each chapter, we indicate the methodology followed: simulation (S), modelling (M), or experiment (E).

the packet header, nodes can detect either if a request for the same file has been sent (aggregation), or the file is within the local buffer (caching). In particular, this latter is a clear example of a concept borrowed from overlay approaches (CDNs, Web Proxies, and henceforth) and fully deployed as a low layer in the ICN context. A caching algorithm for a general network is different (and less simple to analyze and model) from a caching algorithm for a single cache. Indeed, a network caching algorithm is composed by:

- A *forwarding* strategy, for routing requests within the network of caches.
- A *decision* strategy adopted by each cache for determining if the incoming content is worth being cached.
- A *replacement* strategy to decide which element to drop from a (full) cache.

The second part of this Thesis mostly deals with caching in an ICN context: first determining the most influential exogenous and endogenous factors, and then focusing on forwarding strategies. Starting from strategies similar to those analyzed in the HCN section, we show how ICN needs more dynamic forwarding and, above all, it should be tightly coupled with the decision strategy.

1.3 Structure of this work

The structure of this Thesis is synthetically summarized in Tab. 1.2. We report for each chapter the methodology followed: S stands for simulation, M for Modelling, and E for Experiments. We describe in detail the structure of this Thesis in the following:

Host Centric Networking From an HCN perspective, we consider the routing problem within an ISP: this concerns the first part of the work. In particular, Ch. 2 gives a

broader perspective about routing and forwarding into a flat network. Then, with the goal of simplifying the network core, we first develop an algorithm for multiple-path discovery within the boundaries of an Autonomous System (Ch. 3). Then we plug the aforementioned algorithm into a forwarding architecture, namely APLASIA (Ch. 4). In particular, we analytically model the costs of the algorithm,simulatively estimate the routing performance and, finally, experimentally show the forwarding benefits.

Information Centric Networking From an ICN perspective, we consider a particular ICN architecture, namely the Content Centric Networking (CCN) proposed by PARC [56]. As many other ICN architectures, a CCN network can be abstracted as a receiver driven network of caches. Thus, the second part of this work (Part II) is focused on analyzing, modelling, and evaluating caching algorithms within a network of caches. In particular, Ch. 5 provides a detailed background about network caching algorithm, showing how the current literature is not complete, either in terms of proposals or in terms of scenarios. Ch. 6 ranks the factors which mostly affect caching performance. Then, Ch. 7 proposes and analyzes different forwarding strategies within a network of caches. Finally, Ch. 8 describes ccn$\mathcal{S}$im, our simulator tool through which we performed all the simulative work of Part II.

The work ends with Ch. 9 which sums up the major contributions of this Thesis, and indicates the remaining unexplored directions of this work.

Part I

Host Centric Networking

Chapter 2

Background

In the last few years, several research proposals have addressed the issue of decoupling the concept of host identification and location at the Intranet [27–30] or, more recently, at the Internet [31–33] level.

Usually, compact routing [57] represents the theoretical background on which these approaches are based. Compact routing addresses the problem of routing over a network (as we will see, even the canonical hierarchical routing represents a compact routing scheme) by leveraging minimal amount of information (i.e., compact).

Often, these architectures employ traditional routing protocols for their inner-working: as link-state algorithms, such as OSPF or IS-IS, are about 20 years old, they reveal their limits when used in flat environments.

Our goal is to propose a new holistic proposal for the underlying multi-path routing of the above architectures, especially targeting *intra-domain* routing. Our proposal sits at a radical point, unexplored so far, in the network design space, where we tradeoff hardware and algorithmic simplicity for a slightly increased (but tunable) communication cost.

2.1 Aplasia overview

We name our proposal *Adaptive Probabilistic Link-state Architecture Switching in Autoforwarding* (Aplasia). In medical terms, Aplasia refers to a congenital absence of an organ or tissue: in our architecture, this condition refers to the absence of a Forwarding Information Base (FIB) in the switching fabric of core network nodes.

We tradeoff the absence of FIB information with a minimal increase of frame header size: in an Aplasia domain, data frames carry a fully specified source-routed path, so that the next hop interface is always directly available in the frame header (i.e., data frames are *autoforwarding*) and does not require any lookup.

We believe Aplasia to be an advantageous condition. Our reasoning is that data-plane autoforwarding simplifies the nodes hardware architecture (and their cost), as it eliminates

the need to implement FIB data structures able to cope with link speeds and thus requiring fast and expensive memories (e.g., TCAM for longest-prefix match lookup in the IP world, or CAM for Ethernet switching).

Additionally, to further simplify the nodes inner architecture, we reduce the computational complexity of the algorithm run by control plane software (further limiting the hardware requirements in turn), that we again tradeoff for an increased (but tunable) communication cost.

Under link-state algorithms such as OSPF and IS-IS, topology discovery is performed by running the Dijkstra algorithm after each node has received all link-state advertisements. In APLASIA, as the control-messages accumulate the traveled path directly on the header, there is however no need to run additional algorithms for path discovery, that can be instead be learnt and updated on each new control message reception.

The tradeoff between the amount of state needed for routing and forwarding functions, and the quality of the gathered communication path has been explored at all layers of the protocol stack and contexts – from application-layer peer-to-peer overlays, to Internet intra- and inter-domain routing and to lower-layer access architectures at the network edge.

In the current state of affairs, for instance, IP attempts at solving scalability via hierarchical addressing and aggregation. However, hierarchy requires location-dependent names that complicate network management (e.g., mobility) already at the IP level. Location-dependent names can provide enough benefits to become appealing only for specialized environments (e.g., as in PortLand [34], that leverages positional addressing to optimize switching on data-centers arranged as a fat-tree topology). For more general purposes, location-independent (or *flat*) names have received a growing interest lately (see [33] for a thorough overview).

While in principle such flat identifiers can be arbitrary bit-strings, Ethernet MAC addresses represent perhaps the best example. Indeed, as Ethernet is the most widely deployed fixed access technology as of today, much research effort focused on making Ethernet scalable, with relevant research proposals (such as SmartBridges [27], Rbridges [28], Viking [29], SEATTLE [30], SPAIN [36]) and normalization effort grouped under the "carrier grade" Ethernet umbrella [58] (of which examples are IEEE 802.1ah PBB [59] and its traffic engineering extension IEEE 802.1Qay PBB-TE [60]).

We summarize and compare relevant related effort in the above areas in Tab. 2.1, where we list some among the proposal for scalable routing on flat identifiers from an Internet [31–33] or Ethernet perspective [27–30, 36] (the latter being closer to our work given our focus on intra-domain routing). We point out that while each of these two domains has its own specificity, architectural solutions can be shared to some extent. Let us focus on *forwarding state scalability* first. For instance, in the Ethernet context, one of the issues concerns the possibly very large number of hosts – which affects the amount of state that switches have to keep on the one hand, and the procedure used for the host resolution on the other hand. In Ethernet, the space of existing solutions ranges from MAC address encapsulation (as in IEEE 802.1ah PPB, so that only the outer MAC addresses,

Architecture	**Host resolution**	**Routing**	**Algorithm complexity**	**Communication complexity**	**Multipath**
Rbridges [28]	Centralized	LS (OSPF)	$O(NlogN + N\delta)$	$O(N\delta)$	No
SEATTLE [30]	One-hop DHT				No
ROFL [32]	Chord DHT				No
SmartBridge [27]	Centralized	Diffusing computation + BFS	$O(N\delta)$	$O(N\delta)$	No
BANANAS [31]	n.a.	LS + k-shortest path [35]	$O(N\delta + NlogN + kN)$	$O(N\delta)$	Yes (k)
Viking [29]	n.a.	Centralized + multiple runs of [35]	$O(N^3logN + N^3\delta)$	n.a.	Yes (2 out of k)
APLASIA	n.a.	APL	$O(N\delta)$	$O(N\delta)$	Yes (2)

Table 2.1: Comparison of related effort: we show for each architecture its underlying (host resolution and routing) algorithms, and its (computational and communication) complexity bounds.

but not the inner host MAC, are exposed within the domain), to the use of distributed hash tables (as in SEATTLE [30] using a One-hop DHT to perform scalable host resolution). Similarly, ROFL [32] applies techniques from DHTs, though it targets Internet intra- and inter-domain routing.

Let us now consider the issue of *routing and path quality*. In the case of Ethernet, another long studied issue concerns the limits of the Spanning Tree Protocol (STP), whose main purpose is to provide loop free paths between any two nodes in the LAN: here, research has focused on a more efficient use of LAN resources (e.g., by the use of multiple spanning trees) and a faster convergence in case of topology changes or failures. To get around these limits, again the explored design space is rather wide, with solution ranging from centralized approaches [29], to the use of IETF GMPLS [61] control plane protocol suite to configure PBB-TE devices, or the use [30] of classical link-state routing protocols such as OSPF. Yet, even in this case some commonalities can be identified across domains of application, as at the Internet level ROFL [32] still assumes an underlying OSPF-like protocol to detect link and node failures, while Disco [33] resorts to a Distance Vector (DV) algorithm to propagate addresses.

Aplasia operates at an unexplored point in the routing-state vs path-quality tradeoff, as it shifts the routing state from within the core network devices to inside the header of messages traveling in the network. In this work, we do not focus on the name resolution issue that can be handled, e.g., as in [30]. Rather, we focus on two system aspects, namely forwarding and distributed routing protocol, that are necessary to enable the state shift.

2.2 Routing

Broadly speaking, routing algorithms are classified as Distance Vector (DV) or Link State (LS) approaches. In traditional LS algorithms such as OSPF and IS-IS, a full knowledge of the network topology is however needed, which is gained by broadcasting local information among neighbors, after which well-know algorithms such as Dijkstra can be run to compute the shortest path to any given destination (or more complex algorithms to gather multiple paths). In DV algorithms, routing tables are updated incrementally at the reception of each message carrying global reachability information, while loops are resolved by diffusing computation [62] and multiple paths are also possibly supported [63].

Routing in Aplasia follows an Adaptive probabilistic link-state (Apl) algorithm, where nodes propagate, as in LS, local connectivity information. At the same time, Apl performs incremental distributed multi-path computation, and thus inherits some of the desirable DV properties. In Apl each message accumulates the complete traveled path from the advertiser, so that paths can be updated at the reception of any new control message (as in DV) with a greedy algorithm. Moreover, Apl supports the computation of multiple paths, via simple inspection of Apl frame header and comparison operations (unlike in LS), requiring only a minimal amount of state.

While it is possible to limit the number of messages exchanged by Apl to match LS message complexity (that already allows to learn multiple paths), however Apl benefits of some additional exchanges (to ameliorate the quality of the additional paths). These extra messages are (probabilistically) sent in such a way that the Apl procedure is not only auto-terminating (as in DV) and rapidly converging, but the number of messages sent over the whole network remains of the same order of magnitude (with a fixed, tunable, multiplicative overhead with respect to LS). The detailed description of Apl is provided in Ch. 3.

Probabilistic decisions are also used in [64], that employs distributed Ant Colony Optimization (ACO) algorithms, based on swarm intelligence. A set of ants is spread through the network in order to discover disjoint multiple paths, and the pheromone left by the ants is employed in order to probabilistically avoid already crossed paths. However, beside the increased algorithm complexity, we point out that the amount of (pheromone) state kept at each node scales as $O(N^2)$, whereas $O(N)$ in Aplasia[1].

Tab. 2.1 reports the computational and communication complexity of some relevant related work. Given a graph $G = (V, E)$, let us denote with $N = |V|$ the number of nodes in the graph and by $\delta = |E|/|V|$ the average degree. For comparison purposes, let us consider a classical LS approach, as this is used in close work [30]: on any topology change, the number of messages sent over the whole network is $O(N\delta)$, while the computation complexity of Dijkstra is $O(NlogN + N\delta)$.

[1]Note that our currently ongoing work suggest that this could be bound to $O(1)$ under certain assumptions that we briefly discuss on the conclusion section.

SmartBridges [27] achieve lower computational complexity but is still limited to a single (shortest) path forwarding. BANANAS [31] supports instead multiple paths: more precisely, after topological information is disseminated with an LS algorithm, nodes in BANANAS run a k-shortest path algorithm [35], that has a well known computational complexity equal to $O(NlogN + N\delta + kN)$ for finding k shortest paths to each of the N destinations.

Unfortunately, as paths found by [35] are however *not* guaranteed to be disjoint, this may not be sufficient neither for load balancing, nor for resilience. A solution to this problem comes from Viking [29], that however adopts a centralized incremental approach. The central node has first to (i) run a k-shortest path algorithm N times (one per each node), gathering k paths for all $(N-1)$ destinations, with computational complexity $O\left(N(NlogN + N\delta + kN)\right)$; then (ii) for each $N(N-1)$ source-destination pairs[2], it then needs to run a k-shortest path algorithm on a modified graph[3], each of which has a cost $O(NlogN + N\delta + k)$. Overall, the computational complexity for the central node amounts to (i)+(ii)$\approx O(N^3logN + N^3\delta)$ to compute a set of candidate paths (out of which only two are then selected). As reported in [29], this possibly leads to large execution times, on the order of tens to hundreds of seconds, on a 8x8 grid even when only 4 out of 64 possible destinations are considered. Also SPAIN [36] points out that it is "computationally unfeasible to find the best path set of size k". Yet, the solution proposed in SPAIN is again centralized, though greedy and thus less computationally intensive w.r.t [29].

Aplasia aims at finding multiple, as disjoint as possible, paths with a greedy, simple and distributed approach. In this part, we therefore compare the *quality* of the path found by Apl against the centralized optimum computed as in Viking. However, as Viking follows a centralized approach, it does not make sense to directly compare computational and message complexity – rather, we take a classic LS approach as a term of comparison. As shown in Ch. 3, the communication complexity of Apl remains $O(N\delta)$. It follows that, in case only a *primary* and a *secondary* paths are maintained, only $k = 2$ comparison operations are performed for each control plane message, so that the computational complexity per node remains $O(N\delta)$ as well.

Multipath routing is not necessarily related to a specific architecture. Indeed, the set of works about generic multipath routing is quiet broad. [35, 65–69] represents only a small subset of the multipath literature. The multipath problem is often treated jointly with traffic routing in the data plane, solving a multi-commodity flow problem [65] where, given a traffic matrix and a topology, the objective is find the routing that minimizes

[2]Notice that Viking consider that only an subset of nodes $N_S < N$ can be part of a source/destination pair, whereas the remaining $N - N_S$ nodes only perform switching functions and thus should not be accounted for in the algorithm complexity. While this also the likely application scenario for Aplasia, we however prefer to give computational complexity bounds for the general case.

[3]In the modified graph, rather than removing the links, the cost along the k shortest paths is increased by the original graph diameter: in this way, overlaps are tolerated only when strictly necessary, i.e., when the path would otherwise be disconnected.

network congestion. Another class of work closest to ours focuses instead on control-plane topology discovery [35, 64, 66–69]. The simplest link state extension to multipath is Equal Cost Multiple Paths (ECMP) [66], that splits traffic on different shortest paths (if there exist) between each pair of nodes. Other works relax the hypothesis of equal cost, and develop algorithms to find k shortest paths on any given graph [35, 67]. Addressing multiple different paths (i.e., not necessarily the k shortest ones), [68] proposes a link state algorithm, modifying the downstream criterion to avoid loops. Notice however that, while path computation still faces the delay due to link state advertisement, the computation of alternative paths also adds further complexity beyond the $O(NlogN)$ Dijkstra cost (respectively, $O(E + NlogN + k)$ in [35], $O(k(E + NlogN))$ in [67] and $O(E^2)$ in [68]).

We point out that in all LS-based approaches [29–31] a major drawback is that path computation occurs *after* dissemination of the topological information. Topological dissemination constitutes a first source of delay in the path construction process. Yet, gathering multiple paths can further slow down the process due to the growing computational complexity, which is especially true in case more sophisticated path properties are required (e.g., path disjointness as in [29] with respect to simpler k shortest paths in [31]). The fact that Aplasia allows to gather paths *during* topological dissemination constitutes a first advantage, the lack of a costly FIB update process [39] a second speedup, and the simplicity of the algorithm a last one.

2.3 Forwarding and framing

Aplasia employs source routed paths, with a peculiar path encoding that (i) eliminates the need for forwarding tables altogether, by fully specifying the sequence of output ports directly in the frame header. Another nice property of Aplasia framing is the ability to provide (ii) loop-free paths by design. This is as opposite to what generally happens in distributed algorithms, where loop-free properties have to be enforced by the protocol, and are handled e.g., as in DSR [70] by a TTL field (whose setting is however topology-dependent and may thus compromise the quality of the resulting paths) or as in DUAL [62] via diffusing computation techniques.

Framing constitutes an architectural aspect that recent Ethernet evolutions (such as Q-in-Q, MAC-in-MAC, PBB-TE, etc. [58]) have dealt with. However, these approaches merely define new framing capability to enhance Ethernet scalability, but do not otherwise impact the forwarding function. Moreover, such architectures loose the holistic view of the older technology, and are no longer plug-and-play, requiring configuration via a management or control plane, so that we consider a more detailed comparison out of scope.

It could be argued that the idea of stacking multiple labels has already been used in other contexts, as for instance in MPLS. We point out that the label purpose and semantic drift however significantly from ours. Label stacking in MPLS is indeed generally used for flexibility and to overcome label space scarcity [71]. Hence, the depth of the label stack is

generally limited, and a lookup table is still needed to locally translate the label into the corresponding outgoing port.

Closer work under this angle is represented by Pathlet routing [72], Disco [33] and BANANAS [31] in the wired domain, and by Dynamic Source Routing (DSR) [70] in the wireless one. All these proposals go in the direction of specifying the full (or portions of the) path in the packet header, which as stated in [33, 72] introduces low overhead compared to IPv6 headers.

For instance [72] proposes to specify *portions* of edge-to-edge paths (but not full paths) in the header, to extend the routing policy expressiveness in the inter-domain context. BANANAS [31] instead routes along fully specified path sequences (composed by globally known nodes and interfaces identifiers) that are however compactly represented by fixed size *hashes*. During forwarding operation, a local table lookup is still needed to map local link indices to global link ID. However, the size of the table at each router is limited to $log_2 d$ as it depends on the local node degree d, but not on the network size N – which in medical terms already constitutes a beneficial atrophy of the FIB, but not a full aplasia.

Finally, in the wireless context, DSR [70] proposes to fully specify paths directly in the packet header. Differences from Aplasia lay here in the fact that, as the wireless medium is generally broadcast, the path can simply be identified as addresses instead of ports. Moreover, to avoid as possible to occupy the wireless medium with route discovery, DSR makes extensive use of route caching at intermediate nodes (and other techniques to automatically shorten the path if a node further away in the unexpended portion of the path overhears a message reaching a preceding relay). Hence, DSR is engineered by considering the wireless medium a scarce resource –antipodean with respect to the abundance of capacity in Aplasia environments– leading to a completely different architecture design. We underline that a multipath extension to DSR is proposed in [73], where authors employ a reactive flooding algorithm, in which the TTL is gradually increased in order to find more than just one single response. However, while gradual TTL increase is able to provide multiple paths and is more robust to wrong TTL settings, it brings two shortcomings, namely a (i) slower convergence of the routing process, and a (ii) higher number of control messages.

2.4 Part I structure

This first part is organized as follows: in Ch. 3 we provide an overview of Apl , which represents the Aplasia's baseline algorithm: we detailedly analyze both Apl complexity and performance.

Then, in Ch. 4, we plug Apl into Aplasia. We extend Apl to the case of dynamic environments. Finally, we describe Aplasia's forwarding part, which we evaluate by the means of a Click testbed.

Chapter 3

APL : probabilistic routing algorithm

As introduced in Ch. 2, scaling compact routing to very large networks [74] let enterprises deal with simpler and less expensive hardware, which translates in lower capital expenditures and management costs. Every HCN solution proposed so far tries to get, from one side, a scalable architecture in which packets are routed on the shortest path; on the other side, such architecture should be flat [75], self-configuring and possibly self-healing [34]. Many works address this tradeoff, proposing different routing and forwarding schemes, trying to seamlessly fill the gap between local and wide area protocols: Viking [29], SEATTLE [30], PortLand [34] represent few recent examples of research efforts in this direction.

The routing process of such architectures is usually divided in two parts: the *host routing* process [30] aims at resolving each host to a single switch; then, the *switch routing* process let each core machine learn the best path to the other switches [33]. The latter process, to which we focus in the following, is commonly addressed [29,30,34] with a link-state algorithm (similar to OSPF or IS-IS) implemented at layer-2, and that generally yields a single, shortest, path to any other switch (with the exception of [29] that also keeps a backup tree).

We believe that using a link state algorithm at switch level is not the best choice. First, link state decouples topology distribution and routes computation –in the sense that all topological information needs to be received by all nodes prior that the routing table can be computed– which represents a useless waste of time. Then, notice that the core routing table computation algorithm is represented by Dijkstra algorithm, that results in a $O(NlogN)$ complexity. However, Dijkstra only provides a single shortest path between any nodes pair, while additional paths computation produces even higher computational complexity (though still polynomial).

In this chapter, we propose the Adaptive Probabilistic Link-state(APL) as a simpler solution to the aforementioned problem of switch routing, which takes an alternative ap-

proach for the computation of multiple disjoint paths. Aiming at simplicity, we devise a distributed greedy algorithm that drops computational complexity still remaining (almost) stateless and self-terminating. In our design, simplicity tradeoffs with the communication cost, as APL generates an higher number of messages with respect to link state algorithms. At the same time, we can tune (and analytically bound) the number of messages thanks to a simple parameter that drives the speed of the probabilistic procedure. Moreover, by means of simulation on several topologies (up to 10,000 nodes), we show that, with as few as 2-3 times more messages that a link state algorithm, APL is able to find the shortest path and optimal backup path in more than 90% of the cases. In the remaining 10% of the cases, non-optimality is due to a limited amount of overlap between primary and secondary paths (about 1 node on average).

3.1 Algorithm description

In this section, we focus on description of APL routing algorithm. Summarizing the main differences with related literature, APL sits at a different operational point in the tradeoff between algorithmic complexity vs. control message overhead. Unlike in link state algorithms [35, 66–68], since in our approach the whole traveled path accumulates in the control message as in [73, 76], every message brings useful information for path discovery, that can thus be computed online. At the same time, unlike in [73, 76], our algorithm does not rely on critical parameters such as TTL. Furthermore, the actions that need to be taken to handle each message are simple operations, making thus the algorithm viable on simple hardware. Finally, unlike [64, 69], the algorithm is guaranteed to find the shortest path, and as our simulation results confirm, the secondary path is very often the optimal one found by [35, 67] link state algorithms.

We report in Tab. 3.1 the notation used for describing APL 's and its performance. The table is divided into three parts: the top part shows algorithm's parameters, the middle part system model's, and the bottom part concerns the performance evaluation.

3.1.1 Overview

We aim at designing a distributed algorithm for path discovery, capable of finding multiple, possibly disjoint paths between any pairs of nodes. To do so, each node periodically advertises its presence by means of some flooding procedure described below. Specifically, each node sends an *advertisement* message every τ_a seconds; typical values range from a few seconds to minutes [77].

During the flooding procedure, each relay node adds its identifier to the advertisement messages it receives, so that these messages carry information concerning the whole traveled path. Upon the reception of an advertisement message, a node learns a path from the source of this message, as well as from any intermediate node on this path, as in [76]. Flooding decisions are taken independently by each node, and constitute the core of the algorithm.

Parameter	Meaning
β	Exponential back-off
$n_{i,s}$	Number of times node i has seen node s advertisements
V, E	Vertices and edges set
$\mathcal{L}_{i,j}$	Generic network path between i and j
$\mathcal{P}_{i,j}, \mathcal{S}_{i,j}$	Primary and secondary path between i, j
$length(\mathcal{L}_{i,j})$	Path $\mathcal{L}_{i,j}$ hop length
$\mathcal{L}_{i,j} \cap \mathcal{K}_{i,j}$	$\mathcal{L}_{i,j}, \mathcal{K}_{i,j}$ overlap set
M	Network messages for a single advertisement
C_P, C_S	Primary and secondary connectivity probability
O_P, O_S	Primary and secondary optimality probability

Table 3.1: Algorithm notation: algorithms parameter (top), system model (middle), performance evaluation (bottom).

The main idea is that nodes need to flood a received message *at least once*, so that shortest paths are discovered. Nodes actually need to flood the message *multiple times*, in order to discover further paths beyond the shortest one. The number of flooding decisions is critical with respect to both the quality of the path discovery and the overhead of the algorithm.

A simple option [73, 76] could consist in including a Time To Leave (TTL) field in the packet, so as to interrupt the flooding process when some pre-configured maximum path length is reached. The selection of a proper TTL value is critical in this case: if the TTL is shorter than the graph diameter D for instance, then connectivity cannot be guaranteed; if the TTL is too large, the overhead of the algorithm becomes prohibitive (as the number of relayed messages is exponential in the TTL).

We propose an alternative approach based on *adaptive probabilistic link-state* flooding. Any node receiving some advertisement message from source s floods this message the first time, and floods it with some decreasing probability the following times. Specifically, node i floods an advertisement message generated by source node s over all its links (except the one from which it has received the message) with probability:

$$P = \beta^{n_{i,s}} \tag{3.1}$$

where β is some fixed parameter and $n_{i,s}$ is a counter, stored at node i, of the number of times node i has already received an advertisement originated by node s. The flooding decisions are taken independently on each link, and the counter is reset periodically, as explained later.

Note that node i floods the first advertisement message it receives for source node s since $n_{i,s} = 0$ in this case. As further messages are received, flooding will become exponentially less likely, according to the backoff parameter β. Note also that, if we set $\beta = 0$ APL

produces the same number of messages of a link state algorithm for propagating the network topology. The quality of the path discovery is expected to increase with β, at the expense of larger overhead. However, we shall see that performance is not very sensitive to this parameter, and rather degrades gracefully with parameter misguidance, which makes the algorithm robust and practically interesting.

3.1.2 Primary and secondary paths

Consider a network, modeled as an undirected graph $G = (E, V)$, composed of $|V| = N$ routers, in which any pair of adjacent routers are connected by a single link for simplicity (the algorithm can be easily extended to the general case of multiple links between any pair of nodes). Between any two nodes $i, j \in V$, we are interested in finding a pair of *paths*, i.e., sequences of nodes $i = v_i, v_{i+1} \cdots v_j = j \in V$ connecting node i to j in V. We denote by $\mathcal{L}_{i,j}$ the generic path connecting i to j, and by $\mathcal{P}_{i,j}$ and $\mathcal{S}_{i,j}$ the primary and secondary paths, respectively, returned by the adaptive probabilistic algorithm on graph G [1]. We denote by $lenght(\mathcal{P}_{i,j})$ and $length(\mathcal{S}_{i,j})$ the respective lengths of these paths.

To gauge the quality of the primary and secondary paths found by our algorithm, we need to define target path properties. The primary path is expected to be the shortest path in number of hops; in other words, we say that $\mathcal{P}_{i,j}$ is optimal if it belongs to the set of shortest paths from i to j in G (as there may be several such paths). The secondary path is expected to minimize the similarity with the primary path, $|\mathcal{P}_{i,j} \cap \mathcal{S}_{i,j}|$. Note that this choice reduces the share of faith between these paths, improving network resilience against failures and traffic surges.

To find the optimal secondary path $\mathcal{S}_{i,j}$, we consider a modified graph G' in which the cost of links along the primary path $\mathcal{P}_{i,j}$ are increased by the network diameter [37], and other link costs are unitary. As links belonging to $\mathcal{P}_{i,j}$ are now discarded due to higher cost, running Dijkstra on G' we retrieve a path $\mathcal{S}'_{i,j}$ minimizing the similarity function $\mathcal{P}'_{i,j} \cap \mathcal{S}'_{i,j}$ (notice that since nodes along the primary path are not removed from G', they can be included in $\mathcal{S}'_{i,j}$ only if strictly necessary as the path would otherwise be disconnected). We say that the secondary path found by the algorithm $\mathcal{S}_{i,j}$ is optimal if $|\mathcal{P}_{i,j} \cap \mathcal{S}_{i,j}| = |\mathcal{P}'_{i,j} \cap \mathcal{S}'_{i,j}|$ and $length(\mathcal{S}_{i,j}) = length(\mathcal{S}'_{i,j})$, i.e., $length(\mathcal{S}_{i,j})$ of the secondary path is equal to $length(\mathcal{S}'_{i,j})$ of the optimal $\mathcal{S}'_{i,j}$ (as there may be multiple disjoint paths minimizing the similarity with the shortest path).

3.1.3 Pseudocode

A pseudocode description of the algorithm is given in Alg. 1. A source node s initiates the advertisement process by flooding an advertisement packet m to all its neighbors. The flooded packet contains a list of node identifiers ID, initially set to $ID[0]{=}s$ by the

[1] The aggregation of primary and secondary paths at a given node, forms two distinct trees of the network, as in [29].

Algorithm 1: Algorithm pseudocode for a generic node j of the network

```
1  while j is receiving message m do
2      ℓ ← length(m.ID)
3      forall i ∈ [0, ℓ] do
4          if m.ID[i] = j then                                  // loop and abort flooding
5              return
6          else
7              d ← m.ID[i]                                                  // Destination
8              L_{j,d} ← (m.ID[ℓ], ..., m.ID[i])                       // Path overhearing
9              if ∄P_{j,d} ∨ length(L_{j,d}) < length(P_{j,d}) then
10                 P_{j,d} ← L_{j,d}                                // Update primary path
11             cond_1 = |P_{j,d} ∩ L_{j,d}| < |P_{j,d} ∩ S_{j,d}|
12             cond_2 = |P_{j,d} ∩ L_{j,d}| = |P_{j,d} ∧ length(L_{j,d}) < length(S_{j,d}))
13             if ∄S_{j,d} ∨ cond_1 ∨ cond_2 then
14                 S_{j,d} ← L_{j,d}                              // Update secondary path
15     append j to m.ID
16     s ← m.ID[0]                                                          // Source
17     forall next ∈ neighbors(j) do
18         if next ≠ m.ID[ℓ − 1] then
19             send m to next w.p. β^{n_s}                        // Probabilistic flooding
20     n_{j,s}++                                  // Update counter associated with source s
```

source, to which each node appends its own identifier. Upon reception of an advertisement packet m, a node learns a (backward) path to the source s and to any intermediate node $d = m.ID[i]$ along the path. In case the receiver j detects a loop (finding its identifier within the ID list), it discards the message and aborts the flooding procedure. Otherwise, it analyzes, and possibly stores, the newly learned path $\mathcal{O}_{j,d}$. Specifically, the primary (and secondary) path is first set if not existent yet. Also, if the newly overheard path is shorter than the primary path $length(\mathcal{L}_{j,d}) < length(\mathcal{P}_{j,d})$, then the primary path is updated with the overheard one. Similarly, if the overheard path has lower similarity than the current secondary path $|\mathcal{P}_{j,d} \cap \mathcal{L}_{j,d}| < |\mathcal{P}_{j,d} \cap \mathcal{S}_{j,d}|$, or if it has equal similarity but is shorter than the secondary $|\mathcal{P}_{j,d} \cap \mathcal{L}_{j,d}| = |\mathcal{P}_{j,d} \cap \mathcal{S}_{j,d}| \wedge length(\mathcal{L}_{j,d}) < length(\mathcal{S}_{j,d})$, then the secondary path is updated.

Finally, after having added its own identifier to the $m.ID$, the node probabilistically floods the m message, with independent decisions per each neighbor (except the node $m.ID[\ell\text{-}1]$ from which the message came), and updates the per-source counter n_s. Not shown in the pseudocode for the sake of simplicity, ties in the secondary path selection are broken at random.

Notice that, we expect (i) messages on the *shortest path* to reach a node *before* messages that take longer paths. This definitively holds in case of homogeneous delay; otherwise, it may happen that (ii) messages traveling along the *quickest path* arrive first, which would

Network	Segment	N	δ	σ_δ	$\Delta[ms]$	D
Qwest	Core	33	5.0	3.1	5.9	5
DTelekom	Core	68	10.4	13.3	17.2	3
Level3	Core	46	11.7	10.1	8.9	4
Sprint	Core	315	6.2	6.9	3.2	13
Geant	Aggr	22	3.4	1.4	2.6	4
Tiger2	Metro	22	3.6	0.6	0.1	5
Random	-	$10^{[2,5]}$	4	≈ 4	1	$[3, 6]$

Table 3.2: Topological properties of the network scenarios.

be then stored as primary path; analogously, (iii) if control messages queue with data-plane messages without priority, the first message could have as well traveled along the *less congested* path. Notice that, by simple priority queuing, case (iii) can be ruled out. Then, notice that (ii) may only happen in networks having links with very long delays: in this case, the message traveling along the shortest path is not necessarily the first to be received. However, one of the subsequent messages will surely have traveled along the shortest path (since due to $n_s = 0$ the messages are flooded at least once), so that the primary path is guaranteed to converge to the shortest (unlike [64, 69]). However, we will deal with the quickest/shortest path problem in Sec. 4.2.2.

3.2 Performance evaluation

In this section, we precisely quantify the APL *overhead* vs *path quality* tradeoff, and that can be tuned through the β backoff parameter earlier introduced, by proceeding as follows. We first evaluate the overhead of the algorithm through a simple analytical model, then, we employ discrete event simulation to validate the analysis and evaluate performance in terms of the quality of discovered paths.

Simulations are carried on with Omnet++ [38] over four different network topologies gathered by mean of Rocketfuel [78], whose most significant properties are summarized in Tab. 3.2. Specifically, Tab. 3.2 reports the number of nodes $N = |V|$, the average and standard deviation of the node degree, δ and σ, and the diameter D of the original graph G.

Note that we consider both real ISP topologies, corresponding to different network segments, as well as a set of 50 synthetic random graphs with $N \leq 10000$ and $\delta = 4$. For the time being, we use homogeneous settings (i.e., constant and equal delay on every link), and no failures happen within the network. The evaluation of more complex (heterogeneous delay, failures, etc.) scenarios is part of our ongoing work.

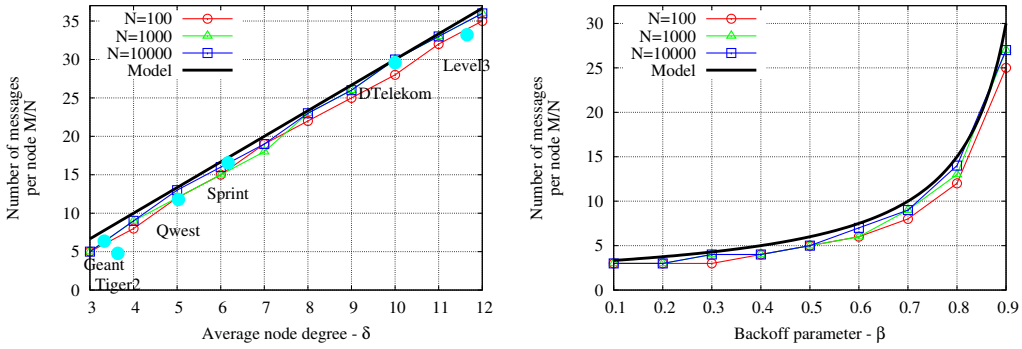


Figure 3.1: Model vs simulation comparison of the number of messages per node for varying node degree δ when $\beta = 0.7$ (a) and backoff parameter β when $\delta = 4$ (b).

3.2.1 Algorithm complexity

We now evaluate the cost of APL in terms of communication, space and computational complexity. The average amount of messages handled by any given node during the advertisement procedure is denoted with M and represents the primary cost sustained by the algorithm. For comparison purposes, we take any link state algorithms like IS-IS or OSPF as a reference. We recall that such protocols are composed by two phases: (i) flooding of topological information, and (ii) path computation, usually fulfilled by the mean of Dijkstra algorithm on the graph information gathered on the first phase.

Communication complexity

The total number of control messages M transmitted over the network during a single advertisement process can be easily estimated neglecting the actual network topology. Considering for simplicity a time slotted execution of the APL algorithm, we have that if node s has started the advertisement process, a generic node j sends $\delta - 1$ advertisement messages (i.e., on every interface except for the interface from which the message came) with a probability Eq. (3.1) that depends on the number of times it previously handled the message from the same advertiser. The total network message count can then be gathered by simply summing up over all possible counter values $n_{i,j} \in [0, \infty]$, and taking into account that all nodes behave the same multiplying by N:

$$\begin{aligned} M &\approx N\left((\delta-1) + (\delta-1)\beta + (\delta-1)\beta^2 + \cdots\right) \\ &= N(\delta-1)\sum_{n=0}^{\infty}\beta^n = N\frac{\delta-1}{1-\beta} \end{aligned} \tag{3.2}$$

Notice that Eq. (3.2) is a conservative estimate of the number of messages, as we do not take into account that loops form and thus messages get discarded. A trivial refinement to Eq. (3.2) would be thus to truncate the sum to N, with a minimal impact as β^n becomes negligible for large n.

Above all, Eq. (3.2) shows that the number of messages M distributed over the network during an advertisement linearly depends on the network size and average degree δ, and hyperbolically on $1-\beta$. While (N, δ) are given by the scenario and cannot be changed in actual deployment, the exponential backoff β gives a very simple knob to tune the overhead with respect to LS. The number of additional messages is off of at most an overhead factor $1/(1-\beta) > 1$, ensuring APL to be scalable as the overhead beyond LS is tunable and bounded.

We validate Eq. (3.2) against simulation in Fig. 3.1, where we normalize the number of messages over the network size to simplify the comparison over networks having heterogeneous sizes. We simulate both the real network topologies early considered (filled circles, to show model accuracy in practice) as well as random graphs with varying degree $\delta \in [3, 12]$ and size $N \in \{100, 1000, 10000\}$ (empty points, to stress test APL). Fig. 3.1 reports the number of messages handled per node M/N during an advertisement round, comparing model and simulation for varying network degrees (Fig. 3.1(a)) and β values (Fig. 3.1(b)), from which is easy to gather in both cases an excellent matching.

Aside the model accuracy, notice that the number of messages generated is very similar for $\beta \leq 0.3$: hence, the path quality results shown earlier for $\beta = 0.3$ can be considered as a lower bound for APLASIA performance. Intuitively, this happens due to the fact that messages are certainly flooded the first time, which already allows to discover more than the primary path. Any further message transmission, for higher values of β, contributes to the refinement of the path quality. However, these refinements may come to the cost of an increase of M, so that alternative techniques (e.g., LoN heuristic) may be preferable than a mere increase of β.

Space complexity

Space needed by APL to run is $O(2N)$ in order to store primary and secondary/backup tree of the network (as in [29]). Additionally, we need also $O(N)$ counters n_s to safely count the number of messages generated by each source. This could be problematic in case the size of the network is not known a priory (though N could be set to a fairly large number for safe operation). At the same time, a staggered advertisement solutions as proposed above, implies that instead of keeping $O(N)$ counters (one for each source in case of advertisements *in parallel*), the system could perform advertisement *in series* and keep a small number of $O(1)$ counters. This could be achieved by desynchronizing the start of each advertisement either with a simple policy (e.g., periodically at random within $[0, 2\tau_a]$) or with more sophisticated schemes (e.g., CSMA-like solutions). This is an interesting direction for future research, that we aim at pursuing in the following.

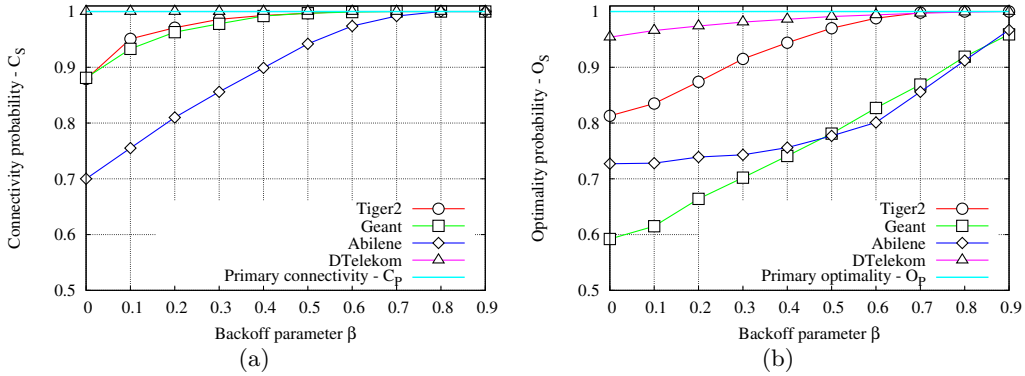


Figure 3.2: APL path quality. We show connectivity (a) and optimality (b) for realistic ISP topologies.

Computational complexity

If Dijkstra and the other graph algorithms need to perform $O(NlogN)$ and $O(NlogN+E^2)$ for the computation of the shortest and backup path respectively once the full graph is known, APL on the other hand needs to perform simpler operations on a packet-by-packet basis. More precisely, on the reception of each control message, switches need to perform: (i) a comparison for the shortest path (Alg. 1, line 9); (ii) an intersection for the best alternate path (Alg. 1 line 12). Since the overall number of APL control messages is bounded, we can bound APL computational complexity as well – which grows with N when all advertisements start at the same time. Instead, it's worth to note that in link state algorithms, a single link state advertisement for a topology change causes to start, for each node, another run of a $O(NlogN)$ Dijkstra algorithm.

3.2.2 Path quality

Let us now focus on the quality of the paths that the adaptive probabilistic flooding algorithm is able to find. For simplicity, we let each node advertise itself once at time $t = 0$ and evaluate the connectivity and optimality of the primary and secondary paths. Since evaluating path quality of random networks is unrealistic, we now only consider the ISP topologies, reporting results over 20 simulations per topology.

We express path quality in terms of *connectivity* along the primary and secondary path (i.e., whether paths $\mathcal{P}_{i,j}$ and $\mathcal{S}_{i,j}$ joining any two nodes $i, j \in V$ exist) and *optimality* (i.e., whether $\mathcal{P}_{i,j}$ and $\mathcal{S}_{i,j}$ are optimal according to the above definitions). We express connectivity in terms of the probability C_P (C_S) that, $\forall i, j \in V$, nodes i and j are connected by some primary (secondary) path. We express optimality in terms of the probability O_P

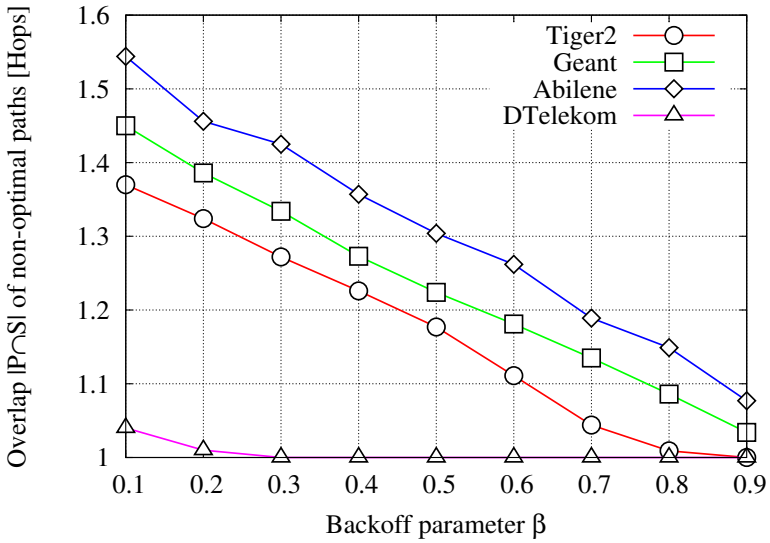


Figure 3.3: APL secondary overlap for realistic ISP topologies.

that the primary path is also the shortest, and in terms of probability O_S that the secondary path is the shortest most diverse path from the primary.

Fig. 3.2(a) depicts the *connectivity* probability of the primary and secondary paths as a function of β: as expected, primary connectivity does not depend on β and is always guaranteed. Since a primary path is always found, the connectivity index is relevant for the secondary path only: we see that all secondary paths are connected in all networks when $\beta \geq 0.7$ (which correspond to limited overhead in Fig. 3.1(b)).

Fig. 3.2(b) reports the *optimality* probability of the primary and secondary paths as a function of β: again, since the shortest path is always eventually found, the optimality of the primary path is guaranteed. Thus, the optimality index is relevant only for the secondary path: we see that a significant percentage (from 60% to 85%, depending on the topology) of secondary paths are optimal even for a very low value of $\beta = 0.1$, and that at least 90% of secondary paths are optimal for all considered topologies when $\beta \geq 0.8$. Moreover, we observe that optimality gracefully degrades β, and furthermore with similar (roughly linear) slope across all topologies. This is a desirable behavior: as no phase transition nor knee appear in the path quality slopes, tuning β between low overhead (low β) vs high path quality (high β) is not critical.

Finally, we dissect the reason behind the sub-optimality of some secondary paths. Recall that a secondary path is optimal if it is the shortest and most diverse path compared to the primary. Hence, sub-optimality of the secondary path may be due to either (i) a non-zero *overlap* between primary and secondary paths, $|\mathcal{P}_{i,j} \cap \mathcal{S}_{i,j}| > 0$, or (ii) a path with a stretch

over the optimal secondary path larger than one $length(\mathcal{S}_{i,j})/length(\mathcal{S}'_{i,j}) > 1$. Fig. 3.3 depicts the overlap, i.e., the number of nodes that primary and secondary paths have in common, conditioning over the sub-optimal paths (i.e., the overlap of optimal secondary paths is not accounted for in the picture). As shown by the figure, sub-optimality seems to be tied to slightly more than one node in common as $|\mathcal{P}_{i,j} \cap \mathcal{S}_{i,j}| \in [1, 1.5]$. Furthermore, as the average overlap is always $|\mathcal{P}_{i,j} \cap \mathcal{S}_{i,j}| \geq 1$ for any β, we can conclude that overlapping paths are significantly more common that long-stretching paths.

3.3 Conclusions

We have presented a novel flooding based algorithm for multiple-path discovery: the algorithm trades a small amount of state in routers, i.e., $O(N)$ counters, in order to significantly limit the number of messages generated by flooding through an adaptive probabilistic algorithm.

Simple analytical bounds, confirmed by simulation results, show the overhead entailed by the advertisement procedure to be low (with respect to the amount of messages needed by classical link state algorithms) and auto-terminating (due to the multiplicative decrease of the flooding probability).

Simulation results also testify excellent performance in terms of path quality: connectivity and optimality of the primary path are achieved by design, while 90% of secondary paths are also optimal when $\beta \geq 0.8$ (or otherwise decrease linearly for lower β). Interestingly, the low percentage of low path is due to a very limited amount of share of faith between paths.

In the next chapter, we use APL as routing component of APLASIA. We analyze its dynamic properties (e.g., advertisement duration, peak communication cost, and hence forth), and then show how APL can be easily coupled with an autoforwarding data plane.

Chapter 4

APLASIA: forwarding on switched paths

In this chapter, we introduce Aplasia, an holistic architecture with a radical design. Aiming at simplifying the inner network devices (and so their cost), we tradeoff node- and algorithmic-complexity for an increased (but tunable) communication cost. The main ingredients of our recipe are (i) the use of complete paths directly in the frames header, that allows core devices to perform data-plane switching functions without lookup and (ii) the use of the Apl (see Ch. 3) greedy probabilistic routing algorithm to quickly discover multiple, near optimal, paths in the control plane. We extensively simulate, analyze and implement our proposal to testify its soundness.

From the algorithmic point of view, we precisely model the duration of a single Apl advertisement round showing as the model fits well simulation results. Finally, some practical considerations are made in order to improve the quality of the paths discovered (e.g., in terms of path disjointness).

From the architecture point of view we accurately design and prototype the forwarding plane of Aplasia. Such design is carried by the practical observation that the original algorithmic solution is well suitable for a source routed forwarding. Additionally, in this work we prototype part of Aplasia concepts in a Click [40] module, and provide experimental results from a small-size testbed.

Aplasia offers a simple, multi-path, flexible, loop-free, fast, efficient, tunable and plug-and-play connectivity layer. To reach these goals, we take an holistic approach and jointly design new *forwarding* and *routing* mechanisms. To simplify the forwarding in an Aplasia domain, *edge-to-edge paths are completely specified in the header* of each frame. Hence, forwarding decisions do not require a table lookup, as the next hop interface is carried in the header.

The remainder of this Chapter is organized as follows. Sec. 4.1 details Aplasia's two main components, namely (i) the autoforwarding data plane and (ii) the Apl algorithm.

We investigate APLASIA performance by simulation (Sec. 4.2), analysis(Sec. 4.3) and experiments (Sec. 4.4). Finally, Sec. 4.6 concludes the chapter.

4.1 Architecture description

APLASIA supports *multiple paths* toward the same destination, that can be used in a flexible way on the data plane. We underline that APLASIA specifies only how to determine multiple disjoint paths between two end points within the AS boundaries. How to exploit these paths (e.g., for backup or load balancing) is left as future research (as briefly discussed in Sec. 4.5). While the underlying APLASIA algorithm does not limit the number of paths a priori, in this work we limitedly consider two paths for the sake of simplicity, and as ultimately selected in Viking [29].In practice, both load balancing and resilience [79] may benefit from the use of multiple $k > 2$ paths.

Furthermore, our *adaptive probabilistic link-state* algorithm yields to a path creation process that is loop-free (by design), fast (quickly providing the primary and secondary paths), efficient (as it very often finds optimal paths) and tunable (in terms of the number of control messages overhead).

Finally, APLASIA operations rely mainly on a single parameter, to whose settings we offer guidance through analytical models, and that simulation results show to be non-critical in case of misconfiguration. Hence, APLASIA can be safely shipped with a default configuration, offering plug-and-play operations as it requires no configuration effort.

4.1.1 Node architecture

APLASIA nodes are addressed by flat identifiers that are univocally associated with them as in [30, 32, 33], and can be chosen irrespectively of nodes topological position.

Whenever a node intends to send data to another node, it needs to assembly a frame, specifying the complete APLASIA edge-to-edge path in the frame header (more details on framing in Sec. 4.1.2). The frame is then handed over to the data-plane for forwarding. As a complete path sequence is specified in the header, the frame is simply pushed to the output interface to which the Current Hop (CH) frame field points to ($CH = 0$ at frame creation, and is incremented by 1 at each hop). Moreover, all nodes along the path perform the same forwarding operation, so that no switching table lookup is performed in the data plane.

Roughly speaking, with reference to Fig. 4.1, we can identify two kind of nodes in the network, namely *edge* and *core* nodes: the former can be traffic sources, whereas the latter only performs switching functionalities. APLASIA follows the principle of pushing complexity to the edge of the network, so that core nodes need to keep only a (i) minimal amount of state (APL counters) to run the adaptive probabilistic link-state algorithm. Additionally, edge nodes store a (ii) cache of source routed paths (e.g., stored as an hash-table or as multiple-disjoint network slices as in [79]). It is worth pointing out that APL

counters are used in the control plane only during the exchange of routing messages and that, similarly, the source-routing path cache is used by edge nodes only at frame generation time. Hence, these structures are accessed at a much slower rate with respect to the data plane forwarding operation deep in the core of the network, where a higher level of aggregation translates into higher speed.

4.1.2 Autoforwarding frames

While providing full details of Aplasia framing specification and encoding is out of the scope, we describe the main fields with the help of figure Fig. 4.1, sketching *data* and *control* frames.

Aside from usual fields such as frame type, node ID, flags, QoS indications and checksum, data frames carry an output path $\{OP_i\}_i$, that represents the set of output ports to follow edge-to-edge in the Aplasia domain, along with a CH pointer to the next output port and a path length field PL. By default, each OP_i consumes 8 bits in the header, which is *optimized* for nodes having at most 256-ports[1].

This choice has a number of advantages, the first of which is to simplify the rest of the architecture by stateless forwarding. This simplicity comes at the price of a (possibly) slightly increased overhead in the frame header, that grows proportionally to the path length. To make a rough comparison, consider for instance the amount of header space devoted to addresses in architectures such as IEEE 802.1ah PBB [59] or paths in BANANAS [31]: in the former, MAC-in-MAC frames carry four 48-bits MAC addresses for a total of 192 bits, while the latter considers 128 bits as "a reasonable bit budget" for encoding the path in the frame header. Sticking for the sake of simplicity to 8-bits individual OP_i, these bit budgets translate to 16-24 hops-long paths. Notice however that while in [31, 59] the 128- or 192-bits overhead is a fixed one, in Aplasia the overhead is instead *variable*, as it depends on the length[2] of the data-plane path.

To sustain the routing operations in Aplasia, the control plane makes use of Apl frames. Instead of merely carrying information about individual links (as in classical link-state algorithms), Apl frames accumulate information about *the whole traveled path*. While algorithms based on diffusing computation [63] already propose to complement link-state exchanges with information about the path (e.g., the second-to-last hop), avoiding loops within the network still remains a fairly complex task. In contrast, Aplasia pushes this trend even further: as paths are fully specified, it becomes *trivial to avoid loops* during path computation (Sec. 3.1).

[1] Aplasia poses no limit to the number of ports per device. An header flag turns on *variable-size* output ports identifiers, whose size depends on a prefix-based notation, inspired by the old classful IP addressing: i.e., 8 bits for identifiers starting with `0` (allowing to address $2^7 = 128$ ports), 16 bits for `10` (allowing to address 2^{14} ports, and used only when the port identifier exceeds 2^7) and so on.

[2] Hence, in practice the overall overhead can be even *lower* with respect to [31, 59], since the path length for some data frames will likely be shorter than 16-24 hops.

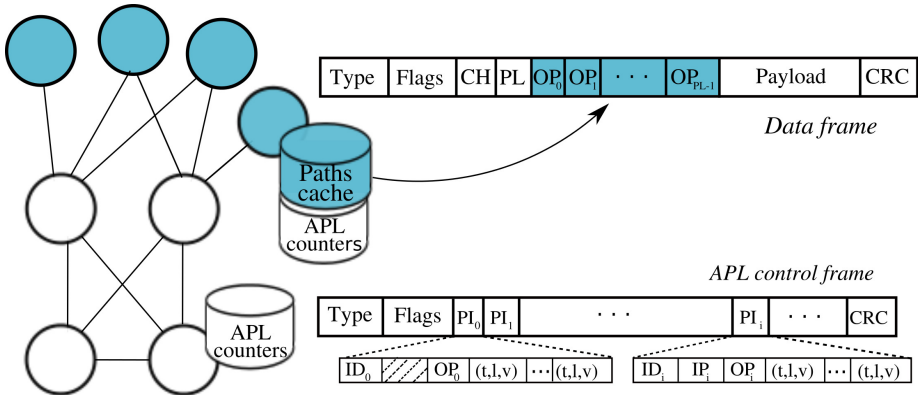


Figure 4.1: Synopsis of Aplasia control-plane routing state and sketch of data and control frame headers.

With respect to data messages, control messages carry additional path information (PI). However, as the volume of control messages is low with respect to data exchanges, the overall control plane overhead is expected to be limited. In more detail, PI_i contains, besides output path OP_i, the corresponding node identifier ID_i (useful for *loop detection*) and input port IP_i (useful for *path inference*). The PI sequence grows at each hop during the path computation process. Additionally, optional information about the estimated path quality can be conveyed in the header, in the form of type-length-value (t, l, v) couplets in PI_i, to assist traffic engineering operations (out of the scope of this work).

As control message carries path information, any edge node, handling or overhearing it, can infer topological information in a completely passive way: more precisely, any node receiving a control message that has already traveled i hops, can in principle learn paths to any of the previous $i-1$ nodes up to the origin, that can (at node will) populate the source routing cache. As different messages possibly carry different information, multiple paths toward the same destination are actually found. Also, while in principle several criteria are possible for primary and secondary selection, for the remainder of this work the primary path is expected to be the shortest path (in terms of hops count). As secondary path, we instead retain the shortest path most disjoint from the primary: as this choice reduces the share of faith between these paths, it improves network resilience against both failures and traffic surges.

4.2 Enhancing path computation performance

In this section, we dissect several aspects of Aplasia path computation. To gather representative performance of Apl , we simulate a single complete advertisement round (i.e.,

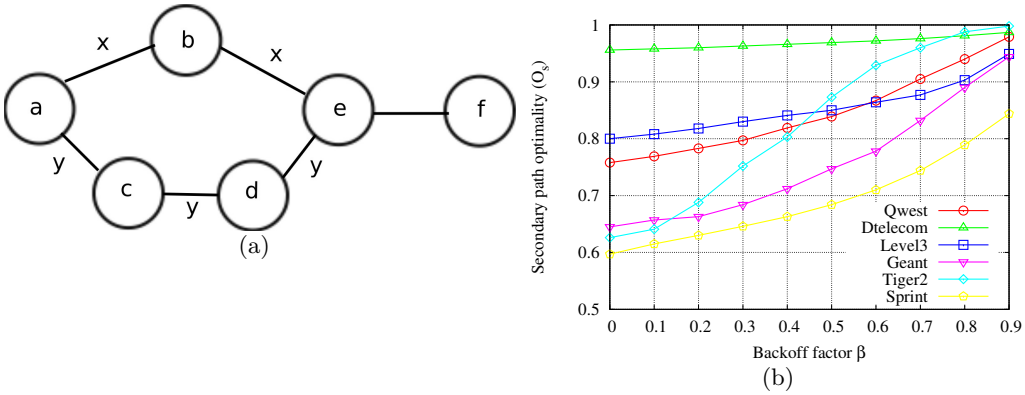


Figure 4.2: Toy case example: quickest vs shortest path(a). Optimality of secondary APL paths(b).

where each node performs a single advertisement), and exhaustively consider all the multiple computed paths interconnecting any two nodes pair in the network. For each parameter setting, results are averaged over 20 simulations runs (i.e., to smooth out different randomized advertisement orders and probabilistic decisions). Notice that we do not expect, in practice, all nodes to learn advertised paths: indeed, core devices performing only switching functionality will likely remain stateless. At the same time, this approach allows to assess quality of the primary and secondary paths that APLASIA is able to find *between any node pair*, thus allowing to get an unbiased picture of APL performance.

4.2.1 Quickest vs shortest path finding

As figured within Sec. 3.1, we expect APL messages traveling on the *shortest path* to reach a node *before* messages that take longer paths: this definitively holds in case of homogeneous delay. Otherwise, it may happen that messages traveling along the *quickest path* arrive first, which could be then stored as primary path. As shown by the toy-case of Fig. 4.2(a), this can happen in networks having links with very long delays. In Fig. 4.2, whenever the $2x > 3y$ condition on the link delay holds, an advertisement from a will reach node e on the path $a-c-d-e$ before $a-b-e$: hence, in this toy case, node f will receive the message having traveled over the shortest path from a with probability β.

In practice, simulation results testify excellent connectivity and quality properties of the *primary* path, as (i) APLASIA nodes are always connected irrespectively of β and of the network topology, (ii) when $\beta > 0$ the shortest path is found in more than 95% of the cases and (iii) the quickest path is selected as primary in the remaining cases.

Similarly, APLASIA nodes are always connected on more than one path. Hence, a more challenging goal is that of finding *optimal secondary* paths, whose quality depends on the

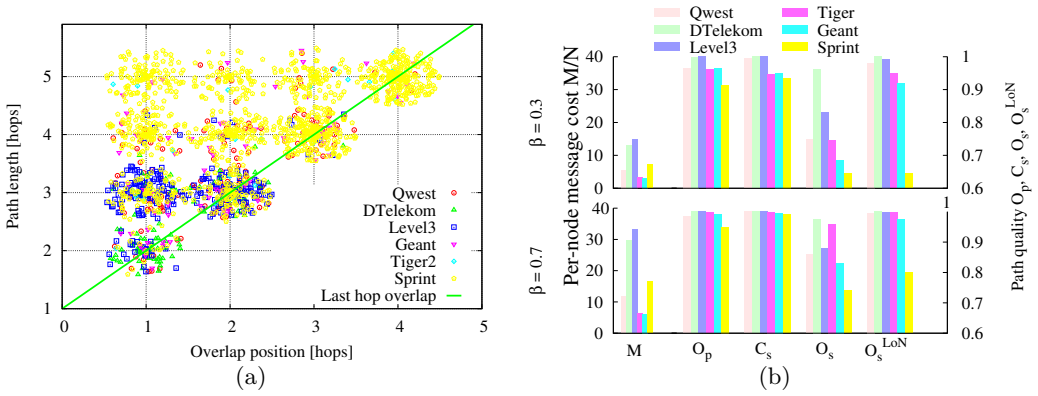


Figure 4.3: Scatter jittered plot of the secondary overlap position versus the length of the primary path(a). Refined performance with LoN heuristic(b).

backoff β. Fig. 4.2(b) reports the probability that the secondary path found by APL is optimal, for all topologies and varying β (we recall that $\beta = 0$ corresponds to the case where APL sends no additional messages w.r.t a classic LS routing algorithm). As Fig. 4.2(b) shows, higher values of β translate into higher quality of the secondary path (e.g., over 80% of the secondary paths are optimal for all networks except Sprint when $\beta = 0.7$), although in case of parameter misguidance (i.e., too low β) the path quality degrades gracefully.

4.2.2 Refining APL

We now pin-point the root cause of non-optimality and propose an effective countermeasure. As pointed out in [80], in most ASs failures happen nearby to the edge nodes (at most two hops far), and rarely internally to the AS topology: hence, paths should be as disjoint as possible n*ext to the edge origin and destination*. Motivated by [80], we cope with this issue close to the source of the advertisement, by letting the advertiser introduce *PI* information for the whole list of its neighbors (LoN) in the advertisement frame. LoN information is useful to nodes whose primary and secondary paths overlap next to the advertiser: as nodes may have gathered paths toward any of the LoN neighbors during previous advertisement rounds, they may decrease the overlap of the secondary path by selecting a more disjoint path through one of the advertiser neighbors.

We report in Fig. 4.3(a) the relevance of the above observation by plotting the position of the overlap in case LoN is *not* included in the advertisement frame. Position expresses the distance in hops from nodes receiving the advertisement to the advertisement originator, so that higher values correspond to overlap next to the advertiser. Fig. 4.3(a) reports a scatter plot of the overlap position versus the path length, using random jittering to enhance the visual presentation. Along the diagonal, we report the percentage of cases

(over all topologies) where carrying LoN in the advertisement message could have helped in reducing the share of faith (i.e., the overlap) between paths.

Fig. 4.3(b) reports, at a glance, the quality of the multiple discovered paths along with the communication cost, for different networks and values of the backoff parameter β. Cost is reported on the left y-axis, expressed in terms of the number of messages M/N handled by each node during a single advertisement procedure (averaged over all advertisements). Path quality metrics are instead reported on the right y-axis: namely, the probability to find optimal primary path O_P, the probability of being connected C_S on the secondary path, and the probability that the secondary is optimal respectively with (O_S^{LoN}) and without (O_S) the LoN heuristic (we avoid reporting C_P as nodes are always connected on the primary).

Notice that the raw number of messages is very limited for both values of β, so that we report $\beta = 0.3$ as an example of Aplasia performance under wrong parameter settings. Then, notice that the shortest path is discovered in most of the cases and that, especially for high β, a secondary path is always found. The quality of the secondary path (i.e., the fact that this secondary path is the shortest path most disjoint from the primary) is instead strongly affected by the LoN heuristic. Notice indeed that LoN affects optimality more than β, as can be easily gathered by comparing O_S^{LoN} vs O_S. Moreover, as O_S value without LoN is already high, the use of the LoN heuristic is able to cope with the remaining overlaps in Fig. 4.3(a), so that Aplasia is very often able to find optimal secondary paths O_S^{LoN} as well. More precisely, with LoN heuristic and $\beta = 0.7$, more than 80% of the secondary paths are optimal over all networks including Sprint (or more than 98% for all networks excluding Sprint).

4.3 Dynamic system performance

We next turn our attention to the temporal properties of the system, examining the duration of the path discovery process, and its auto-termination properties. Since the propagation time is the dominant delay component, we expect time-related properties to be affected by the geographical extension of the network, with possibly wide performance variation across scenarios of Tab. 3.2. Hence, we also develop a model to gather further intuitions on how time-related properties scale with the network size, without being bound to specific topological instances.

4.3.1 Path computation timeliness

First, we assess the path convergence time under Apl , contrasting it with that of classic LS. As it is delicate to directly translate LS computational complexity in a temporal duration (as this depend on the processing speed of the network device), we prefer to resort

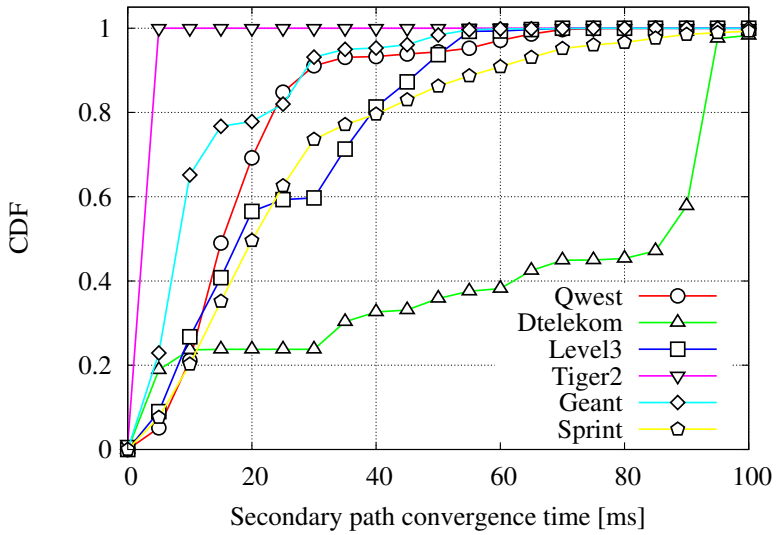


Figure 4.4: CDF of the time employed to converge on the selection of the secondary path (i.e., last execution of line 13 of APL , see Sec. 3.1.3).

to real-world performance of LS algorithms known in the literature [39] for a comparison. Furthermore, [39] analyzes a subset of the topologies we consider in this work (namely, Geant and a large ISP with topology size slightly smaller than Sprint), so that their findings are relevant for our analysis.

As [39] points out, the main source of delay in the convergence of OSPF and ISIS is (i) the FIB update process, with times on the order of *hundreds* of milliseconds, that depend on the processor speed (e.g., GRP, PRP1, PRP2, etc.). The second source is the (ii) execution of shortest path algorithm that requires *tens* of milliseconds at the highest processing speed on large networks (while, clearly, more sophisticated multi-path algorithms such as those used in BANANAS [31] or, worse, Viking [29], could significantly raise the impact of algorithm execution on the convergence time). The third source is (iii) the duration of the flooding process.

We point out that in APLASIA steps (i) and (ii) are skipped altogether and that multiple paths are already available *during* the flooding process, with thus a significant gain in the convergence speed. To evaluate the timeliness of the path computation process we now fix $\beta = 0.7$ as a good compromise between path optimality and extra message cost. Since the primary path is quickly established in APL , path computation converges when a node no longer updates its secondary path (with respect to the algorithm pseudocode, this time correspond to the last execution of line 13). As before, to gather unbiased performance, we let each node start an advertisement, repeating simulation 20 times to average the

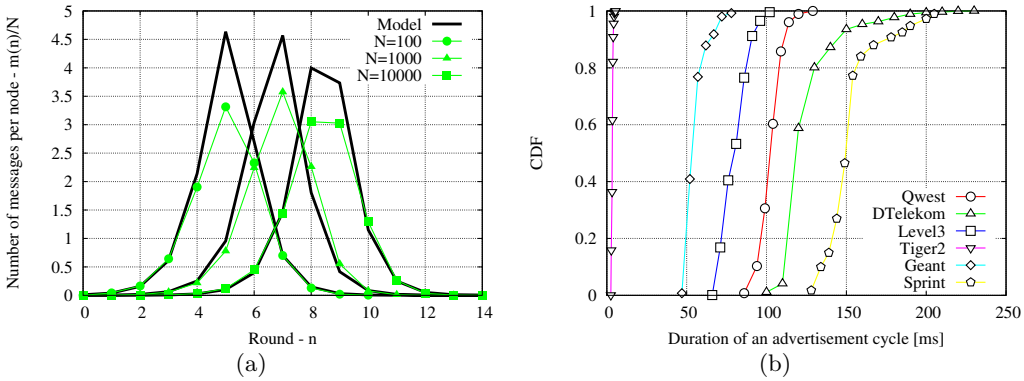


Figure 4.5: Message dynamics (a) and duration (b) of an advertisement cycle.

probabilistic decisions. During each round, we measure the time elapsed between the start of the advertisement at the source and the secondary path convergence at all nodes.

The cumulative distribution function (CDF) of the path convergence time is reported in Fig. 4.4 for all networks. As expected, the raw duration is mostly affected by the average link delay: for all networks except DTelekom (for which the average link delay $\Delta = 17.2\,\text{ms}$ is much larger than that of the other topologies, cfr. Tab. 3.2), 90% of the secondary paths converged in less than 60 ms, and in any case convergence takes less than 100 ms in the *worst case* (while in [39], LS convergence takes more than 100 ms in the *best case*). This confirms the soundness of APLASIA design, that provides fast convergence of multiple paths.

4.3.2 Advertisement auto-termination

We now investigate the temporal evolution of the message propagation during a single APL cycle. With respect to Sec. 4.3.1, we take a complementary view and focus on the duration of the flooding process over the whole network, i.e., before the exponential backoff let the flooding die out.

The objective is not to accurately predict the number of sent messages, which is in any case bounded by Eq. (3.2) in Sec. 3.2.1, but to estimate some critical temporal properties, as for instance the time at which the flooding process reaches a peak, the time at which it vanishes, the impact of the network size, etc.

We consider again a single advertiser and assume homogeneous propagation delays so that time can be considered as slotted. For all $n \geq 0$, the mean number of messages $m(n)$ sent by the N nodes in the network at round n satisfies the approximate recursive equation:

$$m(n+1) \approx m(n)\delta \sum_{k=0}^{\infty} P(k,n)\beta^k, \tag{4.1}$$

where $P(k,n)$ is the probability that a node has received k messages until round n. By convention, the advertiser starts the flooding at round 0 so that $m(0) = 1$.

The approximation in Eq. (4.1) is valid for large graphs, say $N \geq 100$. For smaller graphs, errors in Eq. (4.1) comes both from (i) loops that stop the flooding process and from (ii) the number of edges on which each message is transmitted, which is approximated by δ. Since a message arriving at some node is not sent on the corresponding incoming interface in the actual system, the number of edges on which the message is broadcasted has a binomial distribution with parameters $N-1, \delta/N$: the corresponding mean is $(N-1)\delta/N$, which is close to δ for large N. With respect to our previous notation, we have that $\sum_{i=0}^{\infty} m(i) \approx M$: while M counts the number of messages sent on output interfaces, $m(n)$ counts all messages in flight at a given round (i.e., including the one received from the interface where APL avoids flooding).

Still, the probability distribution $P(k,n)$ in Eq. (4.1) needs to be estimated. We model $P(k,n)$ by noticing that each node has handled $\bar{M}(n) = \sum_{i=0}^{n} m(i)/N$ messages on average up to round n. Since a large number of messages are sent, the distribution of the number of received message is approximately Poisson of parameter $\bar{M}(n)$ at each node, i.e., $P(k,n) \approx e^{-\bar{M}(n)}\bar{M}(n)^k/k!$. Using Eq. (4.1), we deduce:

$$m(n+1) \approx m(n)\delta e^{-(1-\beta)\bar{M}(n)} \tag{4.2}$$

that we can compute numerically. The quality of the approximation is illustrated in Fig. 4.5(a), showing comparison of the numerical solutions of Eq. (4.2) with simulation of random networks, varying the network size up to $N = 10000$ at fixed backoff $\beta = 0.7$ and degree $\delta = 4$ for the sake of illustration (similar matches are gathered for any other parameter settings). For convenience, we normalize the number of messages over the network size $m(n)/N$, to ease the comparison over heterogeneous network sizes.

The model closely captures the bell shape of the message propagation, which confirms that the Poisson assumption holds well in practice. In more details, as shown in Fig. 4.5(a), message dynamics reflect an initial exponential rise due to flooding (as counters are initially 0, hence certainly transmitted since $\beta^0 = 1$). As soon as frame duplicates are received over the network, the exponential backoff kicks in and slows down the message increase, until a peak is reached (at round $n_{peak} = \text{argmin}_n m(n) \geq m(n+1) > 0$), after which the number of messages progressively decreases (and completely stops at about $2n_{peak}$).

From the picture, we gather that n_{peak} increases (i) logarithmically with the network size N, or (ii) linearly with the graph diameter (since in random graphs the diameter scales logarithmically with the network size [81]). This is intuitive, since the flooding slows down as soon as a node starts receiving multiple copies of the message, which is always the case when the path length reaches the network diameter.

4.3.3 Duration of an advertisement cycle

While Eq. (4.2) is useful to show the auto-termination property of APL , it is however not helpful in determining the duration of the process in real networks, for which we resort to simulation. Fig. 4.5(b) reports the CDF of the duration of a single APL advertisement, averaging over 20 repetitions. Comparing Fig. 4.5(b) against Fig. 4.4, we gather that secondary path usually converges earlier with respect to the whole advertisement duration. Moreover, the advertisement still remains short, as it never exceeds 250 ms for the real topologies of Fig. 4.5(b) (nor it would in a $N = 10000$ nodes random graph with 10 ms links delay as for Fig. 4.5(b), since the number of in-flight messages goes to 0 before the 15-th round).

4.4 Click implementation

We have implemented the core APLASIA functionalities in a Click modular router. Our modules fully implement the data plane frame processing and maintenance capabilities, but only partly implement APL control plane functionalities for the time being. To give a rough idea of the implementation complexity, APLASIA Click modules account for about 5000 lines of code, 24 classes and 65 functions.

To summarize the main functional blocks, each physical interface is connected to an APLASIA Port (AP) module, consisting of two distinct Edge Origin (EO) and Edge Destination (ED) sub-blocks. These blocks handle the communication to and from the physical interface, the encapsulation/decapsulation of data from/to the upper layer, the generation of path discovery request/reply, etc. To each AP module is associated an APL module, that assists the AP by duplicating discovery frames to the other ports for path computation. All AP modules of a node are interconnected through a single APLASIA Matrix (AM), whose main aim is to perform the stateless switching function (by inspection of the APLASIA autoforwarding header) and that holds node-wide state (such as the path cache and APL counters).

Primarily, we used the Click implementation for functional verification of the APLASIA principles. In this section, however, we report on preliminary experimental results gathered in a small size testbed, that we use to assess the limits and capabilities of our current implementation.

The testbed is composed by seven PCs equipped with dual-core Intel Xeon E3110 CPUs, clocked at 3.00GHz, equipped with 4 line cards (i.e., four AP and APL blocks each). PCs are arranged as a bus topology, and are interconnected by two point-to-point 100 Mbps Ethernet links[3]. In our setup, the origin node runs in user space (so that it is

[3]In the testbed, we merely use Ethernet cards as point-to-point transceivers between any couple of routers: in other words, no switching, learning or other Ethernet functionalities are used. As a side effect, encapsulation of APLASIA in an Ethernet frame testifies that APLASIA principles are agnostic to the underlying layer.

easy to access timestamping functions without modifying the Click code), while all the others nodes run in kernel mode[4].

We devise two simple yet instructive test scenarios to benchmark the Click implementation, aiming at gathering the atomic duration of (i) data plane forwarding function t_{fw} and (ii) handling of control plane frames during the discovery process t_{cp}. In order for the benchmark to be the least intrusive as possible, we avoid to explicit instrumenting the Click code (e.g., by means of timestamping and `click_chatter` calls). Moreover, we avoid relying on time synchronization, as the precision needed to gather reliable figures exceeds NTP capabilities. Therefore, we decide to infer the atomic cost of message handling in the data/control plane from Round Trip Time (RTT) measurements at different path lengths h. We devise two simple models that account for the data $RTT_{dp}(h)$ and control planes $RTT_{cp}(h)$ respectively, that we later fit on experimental data from our testbed:

$$RTT_{dp}(h) = 2t_g + 2h(t_{tx} + t_{fw}) \tag{4.3}$$

$$RTT_{cp}(h) = 2t_g + 2h(t_{tx} + t_{fw} + t_{cp}) \tag{4.4}$$

In the above expressions, h represents the path length, t_g is unknown value accounting for the message generation time in the AP functional block (counted twice due to the response message), t_{tx} accounts for the message transmission time over the line card (which is known and given by the ratio of the actual frame size over the link capacity). The duration of the stateless switching operation performed by the AM block on a single data frame is represented by t_{fw} (i.e., header inspection and message passing to the right output port), while t_{cp} accounts for messages handling in AP.

We start by determining t_g and t_{fw} by fitting Eq. (4.3) with a first set of experiments, after paths have been established. We carry on experiments for varying path lengths $h \in [2, 6]$, generating 200 RTT samples per h value. In order to measure the RTT, we use an Aplasia OAM tool behaving as the classic ICMP `ping` command of IP networks (i.e., automatically generating response messages). OAM messages have known size, which is accounted for in t_{tx}.

Experimental results are reported in Fig. 4.6, that depicts, for each hop length, the PDF of $RTT_{dp}(h)$ (shaded curves in the left plot) and the first 50 samples gathered during the experiment (points in the right plot). Both plots also show, with a solid black line, the fitting of Eq. (4.3) gathered using an implementation of the nonlinear least-squares (NLLS) Marquardt-Levenberg algorithm. The fit converges to $t_g = 54\,\mu$s and forwarding of $t_{fw} = 45\,\mu$s, with very small asymptotic errors of 0.63% and 0.33% respectively: notice indeed the good match between the model and experimental data (despite a few outliers, possibly due to the user-space device).

We finally perform path advertisement experiments (in a slightly modified setup not detailed for lack of space), to gather t_{cp} by fitting Eq. (4.4) on further experimental data. The

[4]We point out that, as user-space process can be pre-empted at kernel level, we gather conservative results that may slightly underestimate the actual Aplasia performance.

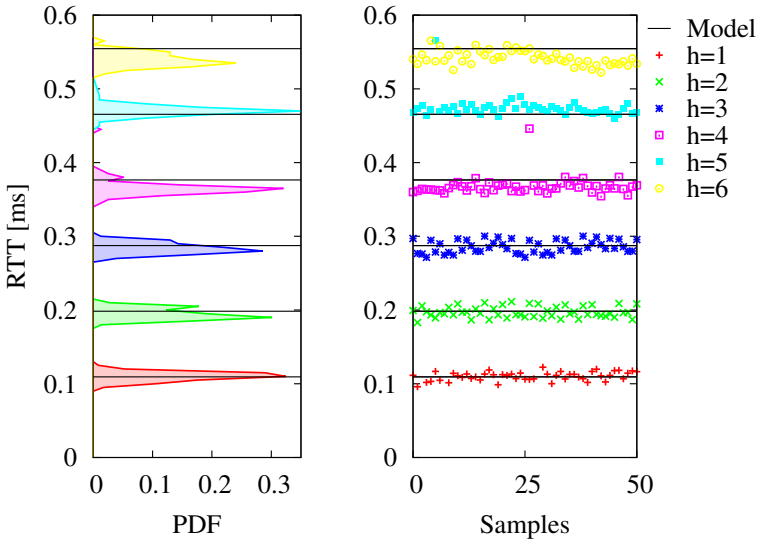


Figure 4.6: Fit of computational cost of data-plane forwarding operation in the Click implementation.

fitting yields an estimate for the control plane overhead of $t_{cp} = 48\,\mu$s (with an asymptotic error of 0.2%), which is thus of the same order of magnitude of the forwarding operation in the data plane. These results further highlight the practical interest of APLASIA and the lightweight of the control plane advertisement task.

4.5 Discussion and Open Issues

This work testifies the overall interest of APLASIA, in terms of control path efficiency, fast convergence time, and path optimality. Notwithstanding, it also leaves some open points, that we discuss in the following and that are part of our future work.

4.5.1 Larger path sets.

While it may be computationally unfeasible to find the *best path set* of size $k > 2$ as [36] points out, it is however possible to simply find a set of k paths. Let consider that the algorithm has just found the primary path (often, the shortest is among the first to be selected). On the reception of a new message, the new path will be stored as the secondary (irrespectively of the path quality, since a secondary is not existent yet). On a further message reception, the latest received path will either be elected secondary (and shifts the secondary in the third position according to some criterion, which is possibly more complex

than merely requiring disjointness from the primary path), or will be appended at the end of the list. Later, when the set of k paths is full, a replacement is possibly needed, and if the decision does not involve comparison among multiple paths, then a greedy algorithm would need to perform (at most) k comparison to take this decision. Hence, the computational complexity of a k-path extensions would remain tolerable $O(N\delta k)$: at the same time, while the quality of the resulting set may be good enough in practice, we acknowledge that this point requires a careful investigation.

4.5.2 Administrative routing weights.

Another open point, that partly goes against the holistic, plug-and-play nature of Aplasia, concern the ability of "emulating" administrative routing weights. In our evaluation, Aplasia treats all link as having equal weight $w_i = 1, \forall i$: yet, we acknowledge that this choice may clash with the current ISP practice of employing administrative weights to bias the construction of the overall topology by specifying arbitrary costs for given links.

An interesting question that remains open is whether it would be possible to tweak administrative weights by using heterogeneous β_i values for different links i (let aside the problem of configuring the individual devices with heterogeneous β_i), and how to map w_i into β_i.

4.5.3 Failure resolution and recovery

While Aplasia finds multiple paths, it does not specify how failure resolution should be handled. In case route cache is stored only at the edge of the network, then in the worst case (i.e., when failure happens near the destination), the recovery time would be on the order of 1.5 RTT (i.e. one RTT to notify the source, plus the one-way delay for retransmission along the backup path).

More efficient failure resolutions could happen in case that (at least some) core devices would be equipped with a path cache: in this case, the message could backtrack toward the source until a node equipped with a path cache is found, that could relay the message on the alternate path (other possible techniques are surveyed in [82]). As [79] points out, by letting traffic to switch between paths at intermediate hops, a source gains access to as many as k^l paths to a destination, with k the number of slices and l the number of switching point on the path.

Notice that the presence of a path cache on a core device would *not* affect stateless autoforwarding in the data-plane under normal operation. At the same time, as now some of the core devices need to be equipped with a FIB, interesting questions would be thus: (i) to explore the tradeoff between the increased capital expenditure versus the improvement on failure handling and (ii) the optimal placement of switching points in the topology.

From the recovery perspective, we should distinguish between stateful and stateless nodes. Clearly, nothing changes for stateless nodes: as they store no paths, no state needs to be updated. For stateful nodes, recovery from failure behaves quite similarly to OSPF [83]. A path computation should be triggered by the node(s) which detects the failure, and the caches are refilled with new discovered paths. In this case, nodes should dispose of fast mechanisms for discarding old and failed paths, substituting new computed ones.

4.5.4 Amount of control plane state

To simplify the description, we assumed that each node keeps a set of $O(N)$ counters to take its probabilistic decision. While this amount of state is scalable with respect to the $O(N^2)$ state required by [64], however it is possible to further reduce the state space requirements. We point out that the main issue is not on the raw amount of state, which is very limited as counters have byte-size (as for practical purposes β^{255} can be considered 0), but to ensure scalability and make a more efficient use of *all* hardware resources, in spirit with Aplasia (and Ockham's razor).

In the normal mode of operation, it will be unlikely for *all* nodes to contemporary start advertisement cycles, though it would be desirable to tolerate some contemporary advertisements. From the evaluation, we know that advertisement has a short duration, moreover slowly growing with the network size. In normal operation it could be thus easy to desynchronizing the start of advertisement operations, by simply employing techniques similar to the Carrier Sense Multiple Access (CSMA) based approach employed in the original IEEE 802.3 Ethernet LANs.

Without entering in details, we have preliminarily tested a CSMA-based approach, requiring that nodes sense whether there is some advertisement ongoing prior to start a new one: in case they receive some advertisement messages during a carrier sense interval, they backoff (as in p-persistent CSMA), otherwise they start the advertisement. The mechanism is imperfect: when other advertisements already have started in some faraway region of the network, but no message attains a node while it performs carrier sense, the latter can start a new advertisement in turn. However, as opposite to destructive interference in medium access protocols, the start of contemporary advertisement do not lead to severe issue provided that the *ongoing* advertisement are counted on a independent counters. In practice, since the whole duration of the advertisement cycle is subsecond, is easy to bound the number of concurrent advertiser (e.g., less than ten with very high probability) by choosing a carrier sense interval on the order of the duration of an advertisement (e.g., on the order of hundreds of milliseconds) – leading to a tolerable delay in normal operation.

As counters need to be reset at each new advertisement cycle, we can implement the counter set as a FIFO queue: when a new control message is overheard, the oldest counter is pushed out of the FIFO, and a new one is inserted and reset. Note that this approach would not only reduces the amount of state to $O(1)$, but could also further contribute in

making Aplasia plug-and-play as the good'ol Ethernet.

4.6 Conclusions

We propose Aplasia, a new, flexible network architecture, requiring simple nodes hardware and little or no configuration. Stateless operation in the data plane is achieved by trading switching tables for header space, as frames carry a fully specified source-routed sequence of output ports identifiers in the frame header. This choice has two important consequences: first, Aplasia frames are autoforwarding, so that core devices need not to be equipped with (nor to maintain) FIB entries. Second, it becomes trivial to ensure that paths are loop-free.

Aplasia supports discovery of multiple disjoint paths through a simple yet effective distributed algorithm, able to discover nearly optimal path at bounded computational and communication complexities. The number of messages exchanged in the control plane remains of the same order of magnitude of classical link-state algorithms, i.e., $O(N\delta)$, as the use of probabilistic exponential backoff limits the overhead to a fixed multiplicative factor $\frac{1}{1-\beta}$. The algorithm requires simple comparison operations at each packet reception, so that the computational complexity is of the same order of magnitude when the set of paths is limited to $k = 2$.

We investigate the main Aplasia performance by means of extensive simulation (on real topologies and synthetic ones up to 10,000 nodes), analytical modeling of main systems aspects, and testbed experiments of our ongoing Click implementation. Our results highlight several desirable properties (such as auto-termination, rapid path creation process, near-optimal quality of the primary and secondary paths, graceful degradation in case of wrong parameter settings, etc.), that testify the overall interest for Aplasia.

Part II

Information Centric Networking

Chapter 5

Background

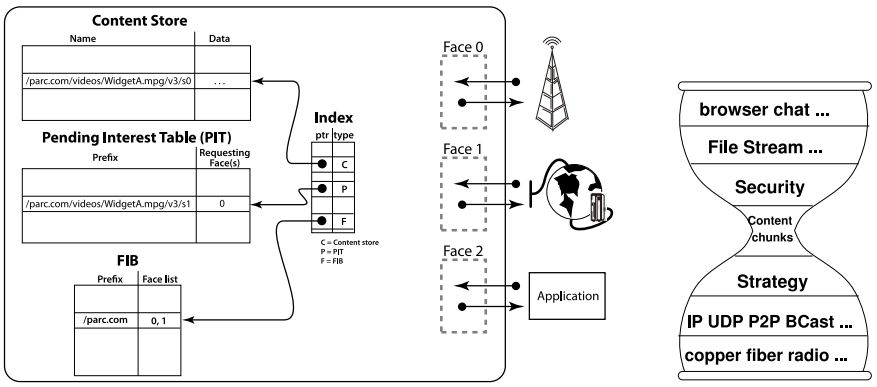
Information Centric Networking (ICN) is a novel network paradigm which places "content" at the thin waist of the hourglass model (see Ch. 1). Despite a large number of ICN architectures have been proposed during the last years (that are overviewed in [1]) nowadays we are still quite far from having operational deployments of ICN at a scale comparable to that of the Content Distribution Networks (CDN) in the current Internet. Prior that large scale ICN deployment will start to rival CDN, a number of challenges need to be solved.

Part of these challenges lay in the design, planning and operation of ICN. While all architectures are unique in some aspects [1], however all ICN proposals agree on the central role that *caching* has in the design – which makes thus sense to consider as primary aspect of ICN. Furthermore, the majority of proposals agree in considering an ICN network as *a receiver-driven network of caches* – from which it follows that practical guidelines concerning the caching aspect that we address in this part will be useful irrespectively of the underlying ICN technology.

At the same time, we also point out that among the many ICN proposals, the Content Centric Networking (CCN) [84] approach has raised significant interest from the scientific community. As such, and since a conceptual unifying framework for ICN is still a work in progress at ICN Research Group (ICNRG) of the Internet Research Task Force (IRTF), in the following we place our work in the context of CCN, and adopt the CCN terminology for the sake of readability.

5.1 Content Centric Networking: an overview

Through this section, we provide a thorough overview of a Content Centric Network (CCN) in order to better contextualize and motivate our work.



(a) (b)

Figure 5.1: Pictorial CCN descriptions: CCN forwarding engine(a) and hourglass(b) (these pictures were directly taken from the seminal PARC paper [41]).

5.1.1 Routing and forwarding

Let's now describe the basic forwarding primitives of a CCN scenario. This section will support the reader through the remainder of this second part of the thesis by introducing the basic CCN terminology. In a Content Centric Network, users ask for contents by the means of *Interests*. The user sends an interest for a given *chunk* of the content she seeks. In this sense, the chunk represents the smallest transfer unit within a CCN network. The interest travels all the network, until it hits either a cache with a *temporary copy* of the chunk, or the repository storing the *permanent copy* of the content. Then, a *data chunk* is sent back toward the client, *consuming* the interest.

We now describe the main data structures used by CCN nodes to forward interests and data chunks. For the sake of description, we report the interiors of a CCN node in Fig. 5.1(a).

- Nodes forward interests by looking up their Forwarding Information Base (FIB) that points, for each content, to the right output *faces*. In CCN terminology a face is a generalization of interface (see Fig. 5.1(a)): for instance, even a simple application socket, sending and receiving interests, can represent a face. Usually, the FIB is filled by a routing algorithm (e.g., [85]), and the set of faces points to one (or more) of the permanent copies of the object. Nevertheless, we will see as the FIB can be updated also with dynamic information (see Sec. 7.3).

- Data chunks are forwarded back to the client by following breadcrumbs left in the Pending Interest Table (PIT) during the interest forwarding phase. In other words,

when the interest gets forwarded by the means of the FIB, each nodes stores the incoming interface within the PIT, discarding interests for which a PIT entry already exists. When the corresponding data come back to the client, these *interested* interfaces are followed back and then deleted.

- Every router is equipped with a Content Store (CS), to temporarily cache received data chunks. When an interest hits the CS, the corresponding data is sent back in reply, if cached in the CS.

The Strategy (or forwarding) layer, sits below the CCN core structures just described (see Fig. 5.1(b)), and basically handles the interest forwarding. As previously said, the FIB contains more than one single interface per chunk. The forwarding layer's task is to choose one among different interfaces available in the FIB to forward the interest it is actually handling.

In figure Fig. 5.1(b) we report the CCN hourglass model. The upper side of the hourglass is meant for application development (e.g., video, web browsing, and hence forth). The bottom side, instead, is implemented by every node of the network, and represents the core of CCN architecture. As in IP, what decouples application from core development is the thin waist of the hourglass, namely Content Chunks and the data structures described above (PIT, FIB, and CS). Indeed, it's worth stressing that CCN architects do not push for a sudden IP replacement. IP is still there in the model. Nonetheless, *if all applications are developed in terms of content data chunks, this will result in a complete shift of the thin waist of the hourglass*, from IP to CCN based protocols.

5.1.2 Naming

In the seminal paper [41] the authors propose a hierarchical name scheme in order to identify the given data chunk. Application inserts the name within the corresponding field of the Interest packet, and sends it over the network.

A CCN name has a hierarchical structure. Each component is separated by a given delimiter. Routing and forwarding is performed on component bases. Note that the naming structure is thoroughly *application driven*. At the lower levels, names represent only a sequence of bytes, used as indexes in PIT and FIB. Thus, the naming choice is completely transparent at the transport layers. Consequently there is not a common naming framework, and each application (e.g., YouTube, Facebook, Netflix, and hence forth) can choose its own naming structure.

For instance, consider the name `/enst.fr/videos/mythesis.mpg`. In this case the (arbitrary) delimiter is represented by the slash, and the name is structured in three components. Being the name a hierarchical entity, FIBs can contain information for only a part of the name, for instance only for the root. Once in the domain `/enst.fr`, FIBs will probably contain more detailed information about the whole data name.

In this sense, routing and forwarding can be quite similar to those of the classical host-centric case (e.g., IP). However, there are two remarkable differences with, for instance, IP routing: on the one hand, the number of first tier domains is much more larger than the number of IP hosts; on the other hand, the CCN names are, by definition, of variable length. Thus, doing an exact matching at FIB level can raise some difficulties (see, for example [47, 86]).

5.1.3 Security

The security mechanisms governing a CCN networks are complex, and it is out of the scope of this thesis describe them in details.

Briefly speaking, under the CCN approach, we shift from securing hosts to securing contents. End hosts retrieving contents from a given repository are to be assured of content validity and trustworthy. This is achieved by signing the whole interest packet with the public key of the publisher. In this way, the publisher binds the name and the content she is publishing. The validation can be done either on a hop-by-hop basis (depending of the router resources) or directly by the end hosts. Keys are represented just as another kind of CCN data, and can be recovered in the same way described above.

Finally, we point out that attacking a single host is not an option within a CCN network. Indeed, the concept of hosts leave space to the concept of content. Denial of Service attacks represent the only type of threats the network could suffer. However, the aggregation provided by the PIT makes data/interest flooding difficult: even though multiple replicas enter within a given router, only a single packet will be actually forwarded.

5.2 Network Caching Algorithms

Having summarily described the CCN architecture, in this section we provide the system model considered throughout the remainder of this work. Besides, we contextualize and compare the model properties with respect to the most recent literature.

5.2.1 Notation

A CCN network may be thought as a *receiver driven network of caches*. Within the following we interchangeably use the terms *request* and *interest*, for indicating the message a user sends in order to retrieve a given content.

We consider a graph $G = (V, E)$ of $|V| = n$ caches of size $C(v)$, $v \in V$, in which aggregate of users connected to a node $v \in V$ request contents at a rate of $\lambda(v)$. Often, we will consider homogeneous scenarios, denoting C the cache size of each router and λ the arrival rate at each client, i.e., $C = C(v)$, $\lambda = \lambda(v) \quad \forall v \in V$.

The catalog $\mathcal{N}$ represents the set of all possible objects i a user can request. The size of the catalog is indicated by $N = |\mathcal{N}|$. Given a content $i \in \mathcal{N}$ we let $p(i)$ be the

content popularity (i.e., the probability that a given content i is requested). Objects may be divided in chunks (as described in the previous chapter). In this latter case, we define $d(i)$ the size in chunks of the content $i \in \mathcal{N}$, and we have $P(d(i) = k) = \frac{1}{D}\left(1 - \frac{1}{D}\right)^{k-1}$, i.e., in the case of chunked contents, the size of each object is geometric distributed, with an average of D chunks.

Each content i is permanently stored within a given repository[1](also said server, or custodian). We denote $\mathcal{S}(i)$ the custodian for content i.

We neglect naming, and security aspects: each content i is represented by its popularity rank (i.e., content 1 is the most popular, and so forth). The popularity for content i is distributed as a Mandelbrot-Zipf: $p(i) = \frac{K}{(q+i)^{\alpha}}$, $K = \left(\sum_{i=1}^{N} \frac{1}{(q+i)^{\alpha}}\right)^{-1}$. α is said the shaping factor of the distribution, since it indicates the distribution slope in a log-log scale. When $\alpha << 1$ the Zipf approaches to a uniform distribution. q is said the plateau of the MZipf, and as more it increases as more $p(i)$ tends to a uniform distribution. In Tab. 5.1 we show the parameter space we investigate in this work.

5.2.2 NCA definition

We define a *Network Caching Algorithm*(NCA) on a network of caches as a triplet $\langle \mathcal{F}, \mathcal{D}, \mathcal{R} \rangle$ of *forwarding, meta-caching* and *replacement* strategies, respectively. Given a node $i \in V$, and a request for a generic file, the forwarding strategy $\mathcal{F}$ has the task to route the request, choosing the right path among those available at node i (e.g., within its FIB). Once data come back (as previously described) node i's decision strategy $\mathcal{D}$ establishes if it's worth caching the current object. In the case of a positive caching decision, a replacement strategy $\mathcal{R}$, selects the element to drop from node i's cache.

The choices considered are listed in Tab. 5.1, and briefly described below. As pointed out in [15], much attention has been devoted to replacement policies $\mathcal{R}$ (over 40 policies are overviewed in [87]), while meta-caching has being studied sporadically (to the best of our knowledge, only by [15,17,19,88,89]). Even less amount of work has been produced in the field of forwarding policies $\mathcal{F}$. To the best of our knowledge, [12–14] are the only work which studies alternative forwarding policies within a network of caches. In the following we describe the NCA literature in terms of forwarding $\mathcal{F}$, meta-caching $\mathcal{D}$, and replacement $\mathcal{R}$.

Forwarding policies - $\mathcal{F}$ Usually FIB are supposed to be pre-filled with routing information toward the nearest repository for each content i. This kind of forwarding is said Shortest Path Routing (SPR). Trough Multipath routing (see Sec. 7.2), the routing process specifies more than one single path per content (either because there are actually more paths toward the custodian, or because there are multiple custodians). Being constrained on the shortest path(s), reduces the possibility of discovering

[1]We can occasionally violate this assumption, providing more replication. Unless otherwise stated, this is the default scenario.

nearest cached copies off-path. In Breadcrumbs [14] the authors propose a forwarding scheme for which data flowing toward the users leaves a trail of breadcrumbs. Nodes decide hop-by-hop if forwarding the subsequent request up toward the repository or down following the breadcrumbs previously remained. In NDN [42], nodes randomly try other interfaces for the same content. In CATT [12], forwarding is done on a per-distance basis: nodes limitedly communicate (by flooding) information about the distance for a given content, and nodes forward request toward the nearest known copy. The final goal of each of the aforementioned policies is to route interests toward the nearest replica (either temporary or permanent) in the network. As such, iNRR asks to an ideal oracle to localize this replica, and NRR implements the oracle abstraction by the means of a scoped flooding (see Sec. 7.3).

Decision policies - $\mathcal{D}$ Generally the assumption is to Leave a Copy Everywhere (LCE), i.e., new content gets always cached. Few exceptions to this rule come from the Web [15,88,89] or CCN [44] contexts: however, the approach in [88] doesn't apply to CCN due to implementation complexity, while DEMOTE [89] is known to poorly perform on network of caches. With this respect, only the Leave Copy Down (LCD) policy [15] is simple enough to be worth implementing in CCN. Again, an even simpler policy is considered by [44], where caching decisions are taken uniformly at random with a fixed probability p. More recently, other decision policies started to appear [17, 19] (not yet published at the submission time), where caching decisions depend either on the position of a node in the topology [19] (expressed by its betweenness centrality) or on the distance traveled by a packet [17]. Finally, we point out that *explicit cache coordination* policies (e.g., see [90] for Web, and [91] for ad hoc domain) would likely violate CCN line of speed constraint.

Replacement policies - $\mathcal{R}$ Least Recently Used (LRU) has been used in the context of CCN [3, 4, 17, 92] and of the more general ICN context [14, 93, 94]. Only few work considers other policies, such as Most Recently Used (MRU) and Most Frequently Used (MFU) in ICN [94]. Still, due to the fact that CCN caching operations must happen at line speed, most of the existing decision and replacement policies are not of practical relevance, as [44] points out. Thus, even a simple LRU policy –often used in turn as a good enough approximation of Least Frequently Used (LFU)– may be too complex to implement, so that uniform random replacement RND may be preferable to keep CCN simple and scalable [44]. We even evaluate a variant of the canonical RND replacement. The BIAS replacement algorithm takes two element at randoms, dropping the most popular. The principle of the BIAS replacement is that more popular items will be more often requested. Thus, choosing the least popular out of two randomly chosen element, increases the probability of finding a popular object within the cache.

Moreover, among the most influencing factors affecting NCA performance, we individ-

Parameter		Meaning	Values(bytes)
Network		Chunk size	10 KBytes
	C	Cache size	10^6 chunks (10 GB)
	n	Network size	
		Topology	Generic (real), Tree, Grid, Torus
Catalog	N	Catalog size	10^8 files(10^{15} bytes)
	D	File size	10^3 chunks (10 MB, geom.)
	$\frac{C}{ND}$	Cache/catalog ratio	$[10^{-5}, 10^{-1}]$
Popularity	α	Shaping factor	{1,1.5}
	q	Mandelbrot-Zipf plateau	{0,5}
Caching	$\mathcal{F}$	Forwarding	SPR,iNRR§7.3,NRR§7.3, Multipath§7.2
	$\mathcal{D}$	Meta-caching	LCE,FIX [15],LCD [16],ProbCache [17], BTW [19]
	$\mathcal{R}$	Replacement	FIFO,LRU,RND,BIAS

Table 5.1: Algorithmic notation and default values.

uate cache and catalog size (Sec. 6.1.1) and content popularity (Sec. 6.1.2). Yet, these factors are constrained by technological limits (e.g., cache size) or determined by the environment (e.g., catalog size and popularity). In this work, a great care is taken to evaluate CCN and NCA under realistic operational points (e.g., as it will be clear on the following, one cannot simply increase the cache size to ameliorate performance).

5.2.3 Scale limits

We now stress what are, in our opinion, limits in terms of scale of the past literature, and why the existing work is not fully faithful of CCN performance in an Internet environment, considering both analytical and simulation based studies.

5.2.3.1 CCN models and simulations

An evident limit of current caching work is that, with few exceptions [3, 4, 44, 47, 92–95], *entire objects* are generally cached. In CCN, content is instead partitioned in sub-objects (or *chunks*), so that different chunks of the same object may end up being cached on different CCN routers. This design decision partly follows from the efficiency of chunk-based diffusion shown by, e.g., BitTorrent in P2P networks, that lately was brought into CDN [96] as well.

Moreover, most of previously developed models rely on different assumptions that are not fit to CCN due to either (i) chunks partitioning or (ii) topological assumption. As CCN requests for consecutive chunks of the same objects are now correlated, the independent reference model (IRM, where all requests are i.i.d.), popular in caching studies, no longer

holds: correlated arrivals are only studied in recent work [3, 4], though the analysis is limited to simple cascade or tree topologies.

Similarly, while authors in [95] provide exact analytic expression for cache hit using either LRU or LFU replacement, and futher consider realistic large scale catalog scenarios, their model still applies to tree topologies, and can thus limitedly represent performance of the access network – but cannot be applied to the network core. Conversely, though the approximate cache model in [8] applies to general network topologies, it considers an object granularity (and further assumes that on a cache miss, the object is instantaneously downloaded before the next request for the same object hits the cache). Hence, to the best of our knowledge, an analytical work overcoming both limits has yet to appear.

Yet, even simulation-based studies of information-oriented networking are often simplistic in their (i) topological assumptions or due to the (ii) scale of the considered system. Indeed, with the exception of [93, 94] (employing synthetic topologies generated with GT-ITM), most simulative work still considers simple topologies (e.g., cascade or trees [3, 4, 44, 92]) and is thus not suitable to investigate multi-path issue, which is in the genes of CCN [84]. Even more important, the scale of the considered system is often underestimated. Notice indeed that, in Web caching it was reasonable to assume pretty large amount of caches (e.g., disks), which implied that an accurate estimate of the ratio between the cache size and the catalog size was not an issue (e.g., add disks). In CCN instead, due to the fact that interest and data packets need to be serviced at the Network Interface Card (NIC) speed, cache size is technologically limited by memory access speed [44, 47]. As it is not possible to arbitrarily increase size of caches on board of CCN routers, it becomes thus imperative to more accurately estimate the size of the catalog that CCN is expected to service, in order to assess whether CCN is able to really achieve the promised breakthrough.

5.2.3.2 Catalog size

The catalog size N can be as low as 250 objects [8], topping to 20K [3, 4] objects for the largest ones. Taking into account also the object size (in chunks) considered in those work, largest catalogs vary between 2Mchunks [4] and 13.8Mchunks [3], i.e., 20GB and 138GB respectively. Yet, we believe these sizes to be extremely small compared to Internet catalogs – which is easily confirmed by summing up the storage size of our portable devices.

We expect system performance to be determined by the ratio of the cache C over the catalog size (in chunks) ND. We point out that this ratio varies in the ICN literature between a minimum of 0.25% [4]-0.5% [94] to a maximum of 10% [8]-20% [94]. Yet, considering realistic CCN cache and Internet catalog sizes, this ratio seems more likely to be on the order of 10^{-5}, hence much smaller than the one considered in previous studies: as a consequence, ICN benefits may therefore be overestimated by current literature.

5.2.3.3 Popularity Model

Another very important aspect that concurs in determining the system performance is the content popularity model. Rather typically [3, 4, 8, 15], CCN studies consider (variants of) Zipf popularity, which is simply tuned by a single parameter α, (i.e., the exponent characterizing the distribution). Yet, no consensus has been reached on the exact model, as for instance [94] resorts to a Mandelbrot-Zipf (MZipf for short) distribution, while [14] also includes uniform popularity, and [3, 4] consider several classes distributed with Zipf popularity (with objects within each class uniformly popular).

Moreover, no consensus has been reached on the actual settings either, as the considered value of α varies in a rather wide range 0.6 [94]-2.5 [3]. For instance, the minimum of 0.6 [94] is taken from a MZipf fitting of Gnutella catalog [97] – where, in this case, the distribution is also determined by the plateau parameter ($q \in [3, 121]$ depending on the Autonomous System under observation, and typically $q \leq 50$). Values of $\alpha \in [0.6, 1.2]$ are adopted in [3, 8, 15, 93], where the extremes are typical of lightly loaded and busy Web servers respectively [98], while large values of α such as 2 [4] and 2.5 [3] derive from the very same analysis of YouTube [48] we used early to determine a realistic Internet catalog size.

Hence, the range of α values are, at least apparently, well motivated. Yet, if we consider more closely the YouTube popularity, for the sake of the example, we see that the $\alpha \geq 2$ value reported in [48] only fits part of the tail, while the body of the distribution appears more likely to follow a Mandelbrot-Zipf law[2].

5.3 Part II structure

The structure of this second part is as follows: in Ch. 6, we start by evaluating caching algorithms over realistic scenarios. In the same chapter we study the impact of the topology on which the caching $\langle \mathcal{F}, \mathcal{D}, \mathcal{R} \rangle$ is deployed. Then, in Ch. 7, we deeply study forwarding strategies $\mathcal{F}$. First, we use the underlying multipath routes provided by an hypothetical routing protocol. Then, we show and analyze strategies which dynamically build the forwarding table, and that require almost no knowledge within router's FIB. Finally, Ch. 8 gives a detailed description of ccn$\mathcal{S}$im, the open source CCN simulator we developed and widely used for carrying on the simulations shown through this second part of the thesis.

[2]We have not access to the popularity data shown in Fig.3 of [48] to perform a fitting, however the dissimilarity from a Zipf shape (and the similarity with Mandelbrot-Zipf) is evident.

Chapter 6

Caching: simulative assessment

In this chapter, we conduct a simulative campaign in order to understand NCA's benefits in a realistic CCN scenario. Among the parameters we explore in this chapters we account the cache size C, the catalog size N, and the popularity distribution. Particular attention has been given to the network structure. Indeed, we analyze the impact of the topology and of its properties. The simulative work has been carried out by the means of ccn$\mathcal{S}$im, a network simulator we developed and fully described in Ch. 8.

6.1 A realistic scenario

In this section, we start by defining a fairly realistic scenario under a CCN perspective. Then, we conduct a thorough simulative campaign in order to rank the main influential factors, affecting the caching performance considering a realistic scenario.

6.1.1 Cache and catalog size

First, CCN chunks are expected to be packet-size (1KB [92]-10KB [3, 4]), hence much smaller w.r.t other ICN architectures (16MB [94]). We select 10KB chunks as in [3, 4]: despite a CCN chunk size is not specified in [84], we believe that in reason of the CCN overhead –due to interest packets and large headers for content naming and security issues– chunk sizes smaller than 10KB would be an overkill.

As we previously argued, the scale of the considered system is often underestimated, or at least the operational point at which the system is evaluated is not always realistic. As far as CCN router cache size C is concerned, both [44, 47] reach the conclusion that 10 GB is about the maximum size that core routers can serve nowadays at line speed. Notice that to forward a packet at line speed, the decision logic only needs to know if the content is cached or not (while the content can be served from a slower, larger, memory). Thus the main bottleneck is the speed at which the index lookup can be performed. Authors in [47] explore the speed/capacity tradeoff of different memory technologies (DRAM, SRAM, RLDRAM,

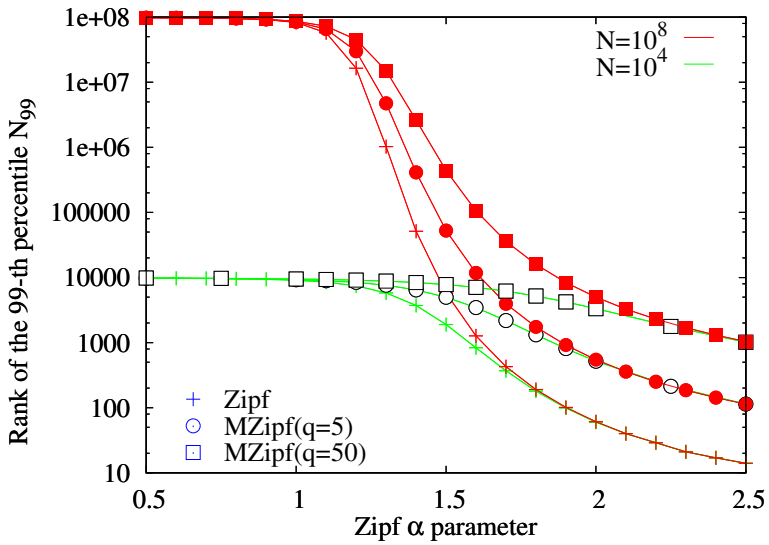


Figure 6.1: Rank of the 99-th percentile of popular requests, modeled according to a MZipf distribution with catalog size $|F|$, exponent α and plateau q.

and so forth). Due to the line speed constraints, [47] concludes that content store indexes should be stored on SRAM: given SRAM size constraint, this in turns caps to about 10GB the size of DRAM content store itself. Authors in [44] instead start from the observation [99] that about half of the caching benefits at packet level happen in the first 10 sec.: considering a capacity of 1Gbps, by employing a two-levels address scheme [44] sizes to about 12 GB the amount of memory that can be addressed at line speed. In spite of these arguments, the cache sizes typically considered in simulation are much smaller (from 6.4MB [92] to 50MB [4] and 2GB [3]), although no proper justification for these selections is given. Hence, we select cache sizes of 10 GB (10^{10} Bytes or 10^6 chunks) as a realistic case study of CCN performance.

We consider a realistic Internet catalog, namely the YouTube portal, due to its wide popularity and growth of video content. Given the approximate number of videos (about 10^8 [48]) and their size (geometrically distributed with average 10MB [46]), the catalog size is on the order of PB (i.e., 10^{15} Bytes) and the cache/catalog ratio is $\frac{C}{ND} = 10^{-5}$.

6.1.2 Content Popularity

Though this might seem, at a first sight, only a minor issue, in reality it is not. The α parameter of the Zipf distribution significantly shapes the request arrival pattern, practically limiting the request to a (possibly very narrow) subset of the whole catalog. Let us evaluate the rank $|F_{99}|$ of the 99-th percentile for different values of the MZipf catalog size

$|F| \in \{10^4, 10^8\}$, and plateau[1] $q \in \{0, 5, 50\}$ parameters, which we depict in Fig. 6.1 as a function of the MZipf exponent $\alpha \in [0.5, 2.5]$. Intuitively, the 99-th percentile rank represents the number of objects in the catalog that make up the 99% bulk of users' requests.

From the picture, it is easy to grasp that for $\alpha \leq 1$, almost the whole catalog needs to be cached to satisfy 99% of the requests *irrespectively of the plateau q parameter*: in this operational region, we expect caching to be hardly effective, due to the unfavorable 10^{-5} cache/catalog ratio early estimated.

Considering the Zipf case, for high values of $\alpha \geq 2$ the 99-th rank converges to the same value *irrespectively of the catalog size*. Moreover, even in the case of very large catalogs consisting of 10^8 objects, 99% of the requests are directed to slightly more than a dozen of objects. Hence, we get that if $\alpha = 2.5$ would hold for the YouTube catalog, a 150MB cache would suffice to store the bulk of the popular objects. Considering the MZipf case, notice that though the curves still converge for high α irrespectively of the catalog size, a higher number of objects needs to be cached to satisfy 99% of the requests (roughly, $|F_{99}|$ grows by one order of magnitude for each step from $q = 0$ to $q = 5$ and $q = 50$).

Hence, we expect CCN performance to be drastically determined by the popularity exponent α (and by the plateau q at a second order). Yet, as there is no consensus so far on the settings of the above parameters, rather than sticking to a specific parameter choice, in the following we explore a wide range of α and q values (as reported in Tab. 5.1) so to assess the boundaries of CCN usefulness. As reference, we will also consider $\alpha = 1.5$ as the centerfold of the $[0.5, 2.5]$ range, and as it corresponds to the phase transition shown in Fig. 6.1.

6.1.3 Performance at a glance

We conduct a thorough simulation campaign, consisting of more than 10,000 simulations exploring over 1,000 individual system parameter settings. In this section, we report the most interesting results obtained from the campaign: our aim is not to provide an exhaustive coverage of our results, but rather to convey a few relevant messages in the most compact way.

To gather performance metrics of interest, we operate as follows. At time $t = 0$ we run the centralized path discovery algorithm, that yields a set of multiple paths between any two node pairs. Starting from empty caches, we simulate the system until caches fill up, at which point we start the collection of all statistics, that continues until the cache hit metric converges to a stationary value. Unless otherwise stated, each simulation point reported in the following represents the average value gathered over 10 simulation runs. By default, we consider a single YouTube-like repository served by CCN routers, to which aggregates of clients are attached, issuing 10 requests per second. We underline that network routers run an SPR forwarding strategy.

[1]Notice that MZipf with $q = 0$ degenerates into a Zipf distribution.

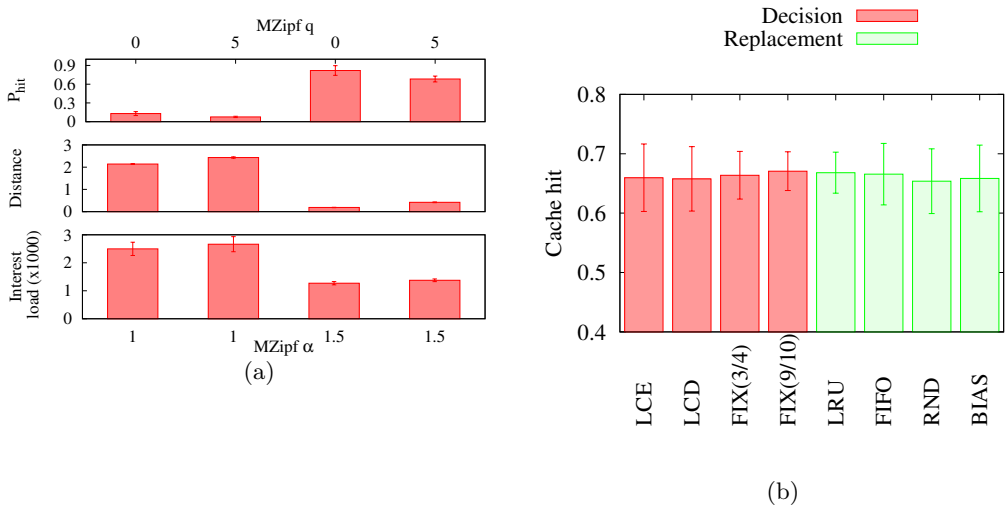


Figure 6.2: NCA performance at a glance: (a) popularity and (b) caching policies. In both figures caches implement an $\mathcal{F} = SPR$ forwarding.

Caching performance is usually expressed in terms of the *cache hit* probability. Additional metrics may be needed in order to capture the network-wide perspective of a NCA. As user centric-metric, beyond the usual *cache hit* probability, we consider the *distance* in terms of the number of CCN backbone hops d that the data chunk has actually traveled in the network (that correlates with user delay). Finally, as network-wide metric, we consider the *interest load*, i.e., the number of links that interests have crossed before hitting a cache. Interest load correlates with CCN router cost (as interest processing must be done at line speed) and with network load (as data is generated in reply to interests, and will flow backtracking the PIT).

Considering a YouTube-like scenario $(D, N, C) = (10^3, 10^8, 10^6)$, we report overall system performance in Fig. 6.2(a) for varying popularity skews $\alpha \in \{1, 1.5\}$, using a basic ⟨SPR,LCE,LRU⟩ triplet. As expected, low α yields to poor system performance, that further worsen for increasing q. It can be seen that *performance metrics are highly correlated*: when $\alpha = 1$, the closest cache does not necessarily have the chunk of interest, hence data travels a longer distance. This in turns lowers the cache hit rate of individual content stores, and translates into a higher interest cost as well. For highly skewed popularity $\alpha = 1.5$ the same content replicates everywhere, so that content of interest is often found in closer caches, hence with a reduced interest cost. We see that α may undermine whether CCN will be able to deliver the promised breakthrough: in case of $\alpha \approx 1$, it seems that a technology change (i.e., the speed and size of memories for content stores) is evidently needed to increase the cache over catalog ratio, and so CCN performance. In the mean-

Table 6.1: Network topologies.

	Segment	$\|V\|$	$\|E\|/\|V\|$	CoV	$\delta[ms]$	D
(a) Abilene	Core	11	2.54	0.19	11.3	8
(b) Tiger2	Metro	22	3.60	0.17	0.11	5
(c) Geant	Aggr	22	3.40	0.41	2.59	4
(e) Level3	Core	46	11.65	0.86	8.88	4
(d) DTelekom	Core	68	10.38	1.28	17.21	3

while, a refined estimation of object popularity may be necessary to evaluate how CCN fits to serve different Internet catalogs (e.g., YouTube, BitTorrent, Netflix, etc.).

Finally, for each routing strategy and all network topologies, we report in Fig. 6.2(b) the average cache hit (i) conditioning over a given decision policy $\mathcal{D}$(and averaging over all replacement policies $\mathcal{R}$, red bars) or (ii) conditioning over a replacement policy $\mathcal{R}$(and averaging over all decision policies $\mathcal{D}$, green bars). Considering shortest path routing SPR with a single repository, this case corresponds to a non-regular tree, where the heterogeneity of link propagation delays further shapes the interest (and chunk) arrival process at the different caches. Yet, rather surprisingly, the performance difference across cache replacement and decision policies is minimal.

We remark that this represents an high-level analysis, useful for providing a relative ranking between different aspects of the system. Furthermore, in this analysis we only consider an SPR forwarding strategy. In Ch. 7 we will show that replacement and decision strategies still have a big performance impact especially if coupled with different forwarding strategies.

6.2 Topology aware caching design

We now evaluate NCA performance considering different topological scenarios, thus determining to what extent the network topology affects NCA performance. We anticipate here that, as recently shown in [2], by using a ⟨SPR,LCE,LRU⟩ triplet, the topology has a minor influence on NCA performance. In this section we remark [2] findings, but then try to exploit the topology design as a means for increase NCA performance.

6.2.1 Caching evaluation on different topologies

To promote cross comparison we resort to real network topologies that are publicly available (some of which gathered through Rocketfuel [100]).

Our selection of topologies is heterogeneous to provide a good span over different network segments. We point out that we consider either metro networks (Tiger2), or backbone/POP networks (Abilene, Geant, DTelekom, Level3). The latter category can be further subdivided in sparse (Abilene,Geant) or more densely meshed (DTelekom, Level3)

topologies. For each topology, we only consider the AS backbone by eliminating all single homed nodes (this is with no consequence, as the last CCN edge router is expected to serve multiple clients). We consider that the system operates at a load well below congestion: hence, as propagation delays dominate transmission delays in the range of expected capacities of CCN routers, we consider infinite backbone link capacity. Tab. 6.1 reports the main characteristics of each graph $G = (V, E)$, namely, the network size $|V|$, the average degree $|E|/|V|$, the coefficient of variation of the node degree CoV, the average link propagation delay δ and graph diameter D.

We make these topologies directly available in the simulator [101], so that alternative approaches can be compared on the very same set of topologies. In a similar spirit, we also consider a standard binary tree topology with 15 nodes and 8 leaves, as typically done in the literature.

Finally, notice that the size of the topologies varies up to a maximum of about 70 nodes (DTelekom). While we argue that these sizes are already non negligible (e.g., since not all nodes in a network may be CCN routers), this also follows from more practical limits of the simulator. Namely, as detailed in Ch. 8 there is a tradeoff between catalog vs network size that can be simulated given a limited RAM memory budget. Instead, we tuned this tradeoff slightly in favor of larger catalog sizes – since they are expected to grow even further.

We inspect the impact of caching policies in Fig. 6.2(b) considering $(\alpha, q) = (1.5, 0)$. We point out that the choice of such a skewed content popularity is not meant for an absolute assessment of NCA performance, but rather for a relative assessment of the impact of several parameters, and is thus perfectly justified.

In reason of wire-speed constraint of CCN, it could be thus worth investigating other, simpler, combination than the most commonly used ⟨SPR,LCE,LRU⟩ pair. Hence, we assess the impact of different topologies considering also random decision and replacement strategies ⟨SPR,FIX($\frac{9}{10}$),RND⟩, yet fixing $\mathcal{F}$=SPR. Fig. 6.2.1 compares cache hit on (i) a binary tree, (ii) the 6 topologies of Tab. 5.1, and (iii) the average over all topologies. Notice that, average ⟨SPR,LCE,LRU⟩ vs ⟨SPR,FIX($\frac{9}{10}$),RND⟩ performance is hardly distinguishable: furthermore, ⟨SPR,LCE,LRU⟩ is *not* always the best choice overall topologies.

Topological information can be exploited at multiple layers, and by multiple planes. In the reminder of this section, we illustrate the case of topology-aware CCN planning. Specifically, we consider several graph-related centrality metrics (e.g., betweenness, closeness, stress, graph, eccentricity and degree centralities) to allocate content store space *heterogeneously* across the CCN network, and contrast the performance to that of an *homogeneous* allocation.

6.2.2 Exploiting topology heterogeneity

In the previous section, we evaluate NCA on arbitrary networks topologies, nonetheless considering homogeneous cache sizes, i.e., CCN content stores have all equal size: $C(v) =$

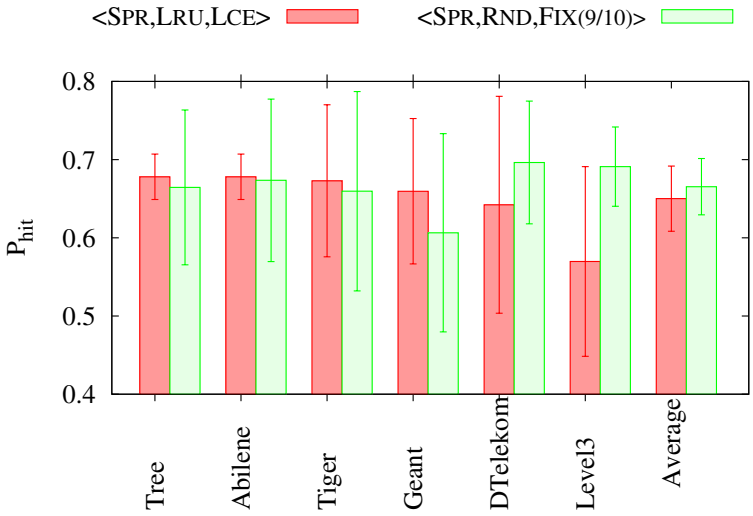


Figure 6.3: Caching efficiency comparison above different topologies.

$C \quad \forall v \in V$. In this section we explore whether *heterogeneous cache size* can improve CCN caching performance. With this regard, closest work to our is [43], that studied hierarchies of Web caches, reaching the conclusion that "stream aggregation (perhaps more than network topology) is a key factor in optimizing cache placement". In [43] the topology is however abstracted by considering several traces, captured at different depths in the end-to-end path connecting a Web browser to a Web server (i.e., end-user client, client proxy after browser cache, network proxy and end-server).

By expanding the work of [43] we compute several centrality metrics over the topology graph G (e.g., betweenness, closeness, stress, graph, eccentricity and degree centralities). We then use the centrality values of each node as a base for different strategies to heterogeneously distribute a given amount of cache, measuring the performance gain (or loss) *relative* to an homogeneous CCN network having the same overall cache amount.
For each of the topology shown above, we compute the following graph-related metrics.

- **Degree Centrality: DC(n)** is defined as the number of links incident upon node n (as we consider bidirectional graphs, we do not differentiate between indegree/outdegree).
- **Stress Centrality: SC(n)** reflects the total number of shortest paths (or geodesics) between all other nodes which run through n. It is defined as $SC(n) = \sum_{\forall s,t \in V \setminus n} \sigma(s,t,n)$, with $\sigma(s,t,n)$ the number of shortest paths from s to t through n.
- **Betweenness Centrality: BC(n)** reflects how often the node n lies on the shortest paths between all the other nodes of the network. It is defined as $BC(n) =$

Table 6.2: Correlation coefficient E[ρ] among centrality metrics.

E[ρ]	DC	BC	CC	EC	GC	SC
DC	1	0.802	0.846	-0.523	0.538	0.903
BC	.	1	0.884	-0.655	0.674	0.949
CC	.	.	1	-0.790	0.794	0.906
EC	.	.	.	1	-0.984	-0.633
GC	.	.	.	.	1	0.652
SC	.	.	.	.	.	1

Note: E[ρ] is averaged over the 6 topologies: the standard deviation is std(ρ) < 0.167 for any metric-pair, and the coefficient of variation is always below std(ρ)/$E[\rho] < 0.310$

$\sum_{\forall s,t \in V \setminus n} \delta(s,t,n)$, where $\delta(s,t,n) = \sigma(s,t,n)/\sigma(s,t)$ is the fraction of all shortest paths between s and t which run through n. In a sense, it is a normalized version of SC.

- **Closeness Centrality: CC(n)** relates to the distance of n to all the other nodes in the network: the lower the total distance toward all other nodes, the more the node is central in the topology. It is defined as the invert sum of the shortest path distances of node n from all other nodes, $CC(n) = 1/\left(\sum_{\forall s \in V \setminus n} d(n,s)\right)$ where $d(n,s)$ is the length of the shortest path from n to s.

- **Graph Centrality: GC(n)** relates to the distance of n to the farthest node: nodes with high GC have short distances to all other nodes in the graph. It is defined as the invert of the maximum of all geodesic distances from a node to all other nodes in the network, i.e., $CC(n) = 1/\max_{\forall s \in V} d(n,s)$ where $d(n,s)$ is the length of the shortest path from n to s.

- **Eccentricity Centrality: EC(n)** reflects how far, at most, is node n from every other node. It is defined as the largest geodesic distance $EC(n) = \max_{\forall s \in V} d(n,s)$ and thus mirrors the GC definition.

The relationship among the different criticality scores is expressed in Tab. 6.2, that reports the average over all topologies of the correlation coefficient between any two sets of centrality values[2]. Though this list is non exhaustive, as other metrics (e.g., the lobby index, etc.) could be included as well, it is nevertheless already fairly representative. It could be rather objected that it is redundant to consider so many centrality metrics, especially since some of them are similar (e.g., BC and SC) or strongly negatively correlated (e.g.,

[2]To simplify the visual presentation, we report only the matrix values above the diagonal: we point out that the correlation coefficient is invariant to the order of the vectors, and hence the matrix is actually symmetrical.

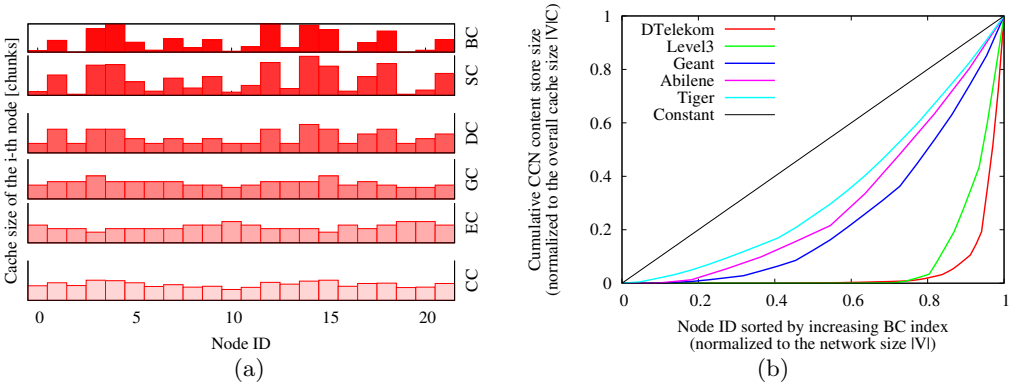


Figure 6.4: Cache size for different ranking metrics applied on the Geant topology (a) and for the same BC ranking applied on different topologies (b).

EC and GC). Still, we point out that each metric has its own pros and cons. Consider for instance EC and GC. On the one hand, it could be argued to give more cache space to high-GC nodes, as they have short distance to all other nodes in the graph, and may thus act as "shared hubs" for content requests passing by. Conversely, it could also be argued to give more cache space to high-EC nodes exactly since they are faraway in the topology: otherwise, their requests may induce higher load on many CCN routers in the path, unless they could be filtered out by having access to a larger amount of cache.

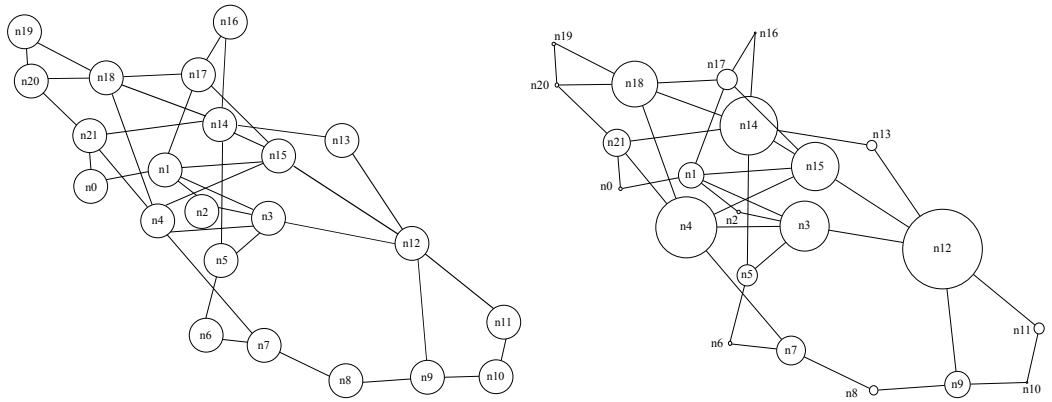
Let us define C_{tot} as the overall size of cache in the topology. In the case of homogeneous network, we fix the size of the individual caches to $Ci = 10\,\text{GB}$, that [44] points out to be about nowadays technological limit due to line speed requirement. In other words, CCN routers must be able to service each interest packet by doing a lookup for content in real time (similarly to IP longest prefix matching lookup for addresses). Hence, memory access speed (and cost) limits the practically achievable content store size [44].

In the case of homogeneous networks, $C_{tot} = |V|Ci$ with $|V|$ the number of nodes in the network. In the case of heterogeneous networks, we exploit the centrality scores as follows. Consider a generic metric $X \in \{CC, GC, DC, EC, SC, BC\}$, where we denote by $X(i)$ the value of X for node $i \in V$ for the considered topology. We then adopt two criteria for cache sizing:

$$C_X^P(i) = C_{tot} \frac{X(i)}{\sum_{j \in V} X(j)}, \forall i \tag{6.1}$$

$$C_X^Q(i) = \max\left(c, \lceil C_X^P(i)/c \rceil c\right) \tag{6.2}$$

Notice that Eq. (6.1) corresponds to a perfectly *proportional* criterion, where the cache



(a) Homogeneous (b) Heterogeneous

Figure 6.5: Pictorial representation of cache sizing, with CCN content store size proportional to the size of node in the corresponding graph. The picture reports (a) homogeneous and (b) heterogeneous cache sizes (in the latter case, proportional to the betweenness centrality BC index).

size $C_X^P(i)$ is distributed to the i-th node proportionally to the metric $X(i)$ normalized over the sum of the $X(i)$ score over the whole graph. An example of the Eq. (6.1) strategy computed for all centrality metrics on the Geant network is reported in Fig. 6.4(a) (where the centrality metrics X are sorted top to bottom by decreasing coefficient of variation of the score $\sigma(X)/E[X]$).

While Eq. (6.1) is an ideal strategy that allows to gauge the relative importance of the centrality score, we acknowledge that it may be hardly feasible in practice: indeed, CCN content store modules will be quantized in multiples of a unit module c, as it happens for nowadays RAM memory. As such, we also consider a *quantized* strategy Eq. (6.2), where the size of individual caches $C_X^Q(i)$ is multiple of $c = 1\,\text{GB}$ units. The model assumed by Eq. (6.2) is that ISPs will invest in a fixed number of memory modules C_{tot}/c that they can then arbitrarily deploy in the network. The viewpoint we adopt is that an ISP may wish to reallocate the C_{tot}/c cache modules at its disposal (e.g., moving from an homogeneous setup to an heterogeneous one), so to optimize the achievable performance without incurring in further capital expenditure.

For the sake of illustration, Fig. 6.5 depicts nodes of variable size, whose radius is proportional to the cache size $C(i)$, to visualize *where* in the network the cache resource have been allocated. Specifically, Fig. 6.5 contrasts an homogeneous Geant network where $C(i) = C_{tot}/|V|$ for all nodes (left plot) with the heterogeneous $C_{BC}^P(i)$ allocation (right plot) corresponding to the largest skew in the cache resource allocation (as seen in the top plot of Fig. 6.4(a)). Clearly, the other centrality metrics will provide allocation skews in

between these two extremes.

Notice that the quantization process induces an error so that the overall cache amount is now $C_{tot}(1+\epsilon)$ with $\epsilon \in \mathbb{R}$ the error induced by the quantization process. This error is due to two contrasting operations[3] in Eq. (6.2) that somehow compensate (the average error is $E[\epsilon] = 2.3\%$). It could be argued that any difference in terms of performance (e.g., cache hit gain) may be due to discrepancy in the quantized cache size (e.g., when $\epsilon > 0$). To rule out this possibility, we verify the absence of correlation between these discrepancies. More formally, denote H_{Const} the cache hit probability of the homogeneous network, with overall cache size C_{tot}. Denote then by $H_X^Q = (1+\gamma)H_{Const}$ the cache hit of an heterogeneous network with quantized cache allocation strategy according to the centrality measure X, with overall cache size $\sum_{i \in V} C_X^Q(i) = (1+\epsilon)C_{tot}$. The correlation coefficient $\rho_{\gamma,\epsilon}$ between the series of (γ, ϵ) pairs gathered over all networks and centrality measures, equals $\rho_{\gamma,\epsilon} = 0.05$, ruling out any correlation between these errors. As such, performance differences in the following solely depend on the centrality metrics.

Notice that Fig. 6.4(a) and Fig. 6.5 highlight variation in the cache allocation according to different centralities for the same topology. In Fig. 6.4(b) we finally report a complementary view, showing the same centrality index (namely, BC) for all topologies. The x-axis represents the normalized node ID $i/|V|$, sorted by increasing BC index; the y-axis instead reports the cumulated fraction of content store size $\sum_{j=0}^{i} C_{BC}^P(i)/C_{tot}$, where we consider a proportional allocation strategy for the sake of illustration. It can be seen that in the DTelekom and Level3 network, BC index defines a very skewed allocation, with few nodes taking the most of the cache. Conversely, Geant, Abilene and Tiger provide a more balanced allocation (approaching the constant allocation depicted as a reference).

6.2.3 Performance evaluation

The aim of our wide-range simulation campaign is (i) to assess whether heterogeneous cache sizing can provide performance benefits over an homogeneous network and (ii) if performance gain are consistent across all topologies for some allocation metric $X \in \{CC, GC, DC, EC, SC, BC\}$. As such, we adopt the simplest strategies namely ⟨SPR,LCE,LRU⟩ for carrying on the aforementioned campaign.

We express caching performance with two output metrics. We consider *cache hit* probability H as the usual network-centric metric. As user-centric metric we consider the *path stretch* $d/|P|$ as the number of CCN backbone hops d that the data chunk has actually traveled in the network, normalized over the path length $|P|$ until the content originator (i.e., without caching). Note that $d = 0$ when users find the content at the edge CCN router and $d = |P|$ when the content is not cached by any CCN router, so that $d/|P| \in [0, 1]$.

First, we observe that *quantization* actually plays a beneficial role. Let us consider the cache hit metric H and as before denote by H_X^P (H_X^Q) the cache hit metric achieved

[3]The max operation imposing a minimum cache size $C_X^Q(i) \geq c \quad \forall X, i$ increases ϵ, while the ceiling operation reduces ϵ.

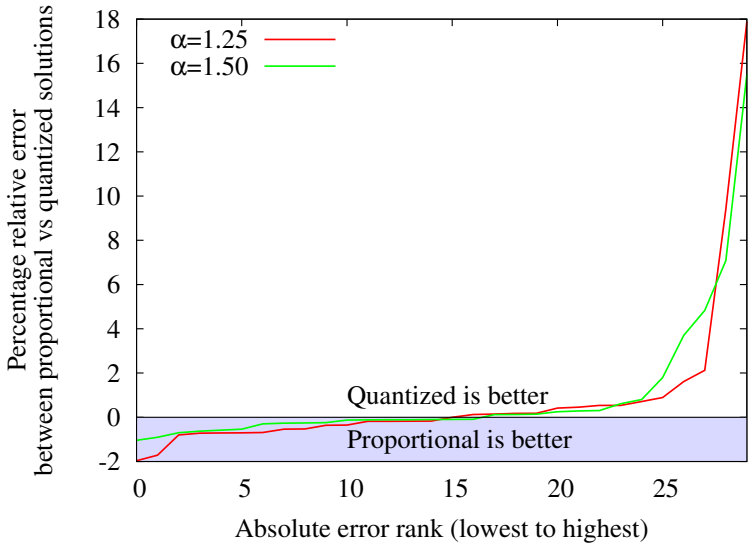


Figure 6.6: Percentage relative error $(H_X^Q - H_X^P)/H_X^P$ for the cache hit metric of quantized vs proportional allocation, all topologies and allocation metrics.

for a given topology and popularity settings by using a proportional (quantized) allocation according to centrality metric X. We then define the relative error induced by quantization on cache hit as $(H_X^Q - H_X^P)/H_X^P$, which is depicted for all topologies and metrics in a monotonously increasing fashion in Fig. 6.6. In the picture, a negative value (gray shaded zone) corresponds to cases (i.e., metric and topology combinations) where proportional allocation would yield better cache hit results. Interestingly, Fig. 6.6 shows that though proportional allocation yields better performance in some cases (left part), however the performance difference with the corresponding quantized allocation is minimal (below 2%). Conversely, there are cases in which quantization can bring almost up to 20% gain with respect to a proportional allocation. Essentially, this is due to the fact that some metrics (especially, BC and SC, recall Fig. 6.4(a)) may allocate a very low amount of cache space to some nodes, which is "corrected" by having a minimum amount of cache in the quantization process. In the following, we thus consider only quantized allocations: this choice is both robust (as we avoid outliers due to skew in the centrality metrics) and realistic (as a perfectly proportional allocation is not directly applicable).

To represent at a glance the impact of network topology and allocation criterion on the system performance, we depict in Fig. 6.7 a scatter plot of the cache hit versus path stretch for different criterion and topologies (reporting only $\alpha = 1.5$ for the sake of brevity). Each box is centered around the average cache hit and stretch performance, while box width and height represent their standard deviations respectively. For the sake of illustration, each

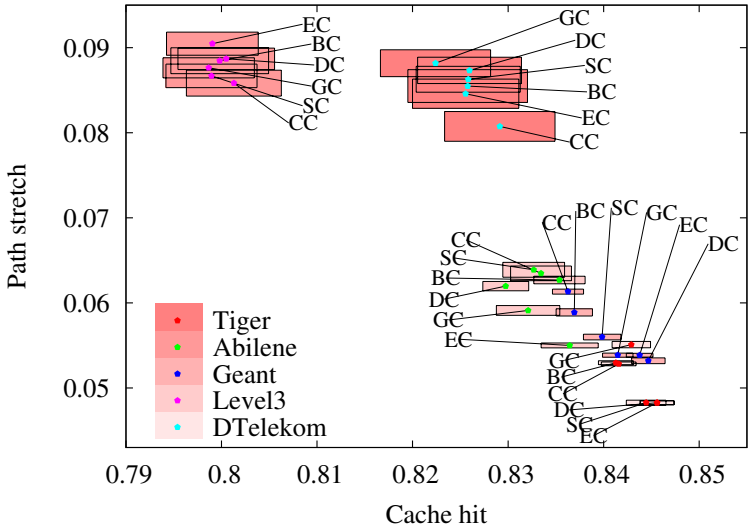


Figure 6.7: Scatter plot of cache hit versus path stretch for different ranking (explicitly labeled) and topologies (represented with different points and box colors) when $\alpha = 1.5$. Each box is centered around the the average hit and stretch performance, with box width and height extending to the standard deviation of the above respective metrics.

topology is represented using different points and box colors, while centrality metrics are explicitly labeled in the figure.

Notice that for Level3 and DTelekom, the cache hit is only modestly affected by centrality metrics. That happens despite a significant unbalance in the network topology, with the vast majority of the cache resources possibly allocated to few nodes only (as per Fig. 6.4(b)). For instance, for Level3 all boxes roughly spans along the same x-axis support, meaning that heterogeneous allocation can hardly make any difference on the cache hit performance. Conversely, a slightly more pronounced separation is visible along the y-axis, entailing that the number of hops traveled before a cache is hit can instead be affected by the ranking function. Similar considerations hold for DTelekom, where performance gaps are only slightly more noticeable.

Centrality metrics have a larger impact instead on the Tiger, Abilene and Geant topologies, for both cache hit and path stretch metrics: notice indeed that boxes now separate across centrality metrics. Due to the cache hit and path stretch semantic, the bottom-right part of the plot corresponds to better performance, so that allocation criteria resulting in bottom-right boxes for all topologies should be preferred. Due to the previously observed correlation in the size of the resulting caches (Tab. 6.2) it is not surprising to observe similarity across allocations of different centrality metrics. Yet, there is no clear winner that

stands out from the plot – such as an optimal strategy, furthermore yielding consistently superior results over all[4] topologies.

We stress the Zipf popularity settings may further alter the results for a given topology. We exemplify the situation by taking into account the relative error between the cache hit gained by a quantized allocation strategy induced by metric X over a constant homogeneous allocation, i.e., $(H_X^Q - H_{Const})/H_{Const}$ according to our previous terminology. The relative error is depicted in Fig. 6.8 for $\alpha = 1.5$ (top) and $\alpha = 1.25$ (bottom) and for all topologies (points) and centrality metrics (x-axis).

It can be seen that, e.g., eccentricity centrality EC may have an opposite behavior depending on whether the popularity is highly skewed or not: apparently, putting more cache capacity to nodes that are somehow "confined" in faraway networks region can be a reasonable choice only for highly skewed content ($\alpha = 1.5$) as it otherwise can yield to cache hit reduction. More generally, even for a single performance metric, it is again hard to find a metric robust under all networks and popularity settings – *with the exception of degree-based* $C_{DC}^Q(i)$ *allocation.*

Notice indeed that, for both $\alpha = 1.5$ and $\alpha = 1.25$, the DC metrics yield to cache hit improvements for all topologies except Abilene, where the performance loss keeps however very limited (i.e., performance are very close to the baseline constant allocation for Abilene). This can be explained with the fact that, in the Abilene topology, the degree does not significantly vary across nodes, so that a degree-proportional allocation practically degenerate into an (almost) homogeneous allocation. Conversely, in all other cases $(H_{DC}^Q - H_{Const})/H_{Const} > 0$ so that some gain can be gathered from a heterogeneous allocation.

Overall, it seems that a simple quantized degree-based allocation policy C_{DC}^Q is the most robust across all topologies. On the one hand, this is positive, since very simple operational rules of thumb can be defined (e.g. "if you add a line card, add a content store module as well"). On the other hand, we also observe that a degree-based selection may tradeoff with technological constraints, such as the ability to perform memory lookups at line speed *when both the memory size and the node network capacity grows.* As a result, further architectural analysis beyond [44] may be needed to assess whether the quantized degree-based allocation policy is also feasible, and thus relevant.

Finally, from a higher level viewpoint, we notice that performance gain is upper-bounded by a modest 2.5% *in the best case* (Level3 topology, $\alpha = 1.25$ in the bottom plot of Fig. 6.8). In reason of such a limited gain over homogeneous cache sizes, it seems as there may be no real incentive in using heterogeneous cache allocation policies altogether.

[4]We prefer to ensure that gain hold over a larger number of topologies that simply requiring a larger *average* gain, as the latter may hide performance drops for some topologies.

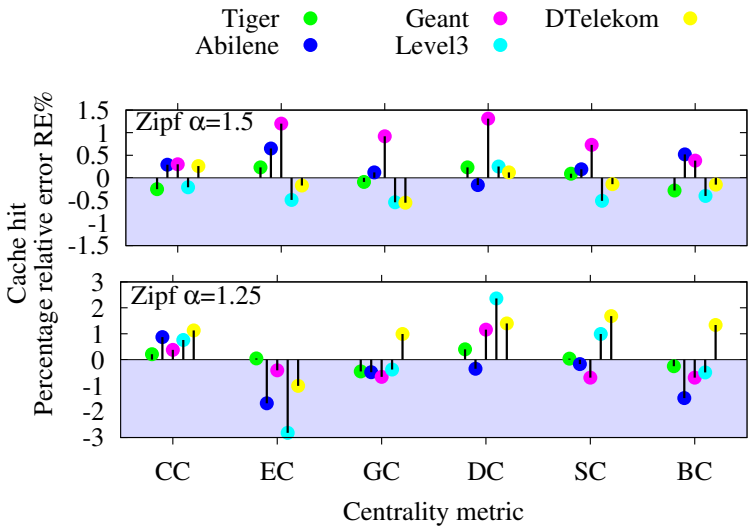


Figure 6.8: Cache hit under quantized ranking allocation strategies, for different topologies and Zipf settings. Y-axis reports the percentage relative error $(H_X^Q - H_{Const})/H_{Const}$ with respect to a baseline constant allocation. Dark shaded zone corresponds to a performance loss for heterogeneous allocation.

6.3 Conclusions

In this chapter we assess which factors most influence a network caching algorithm. We gather that (i) catalog and popularity settings play by far the largest impact in determining NCA performance. While this is an expected result, it also means that much remains to be done in order to fully assess whether CCN can deliver the promised breakthrough. The lack of common evaluation scenario is a critical point that the ICN community should address in the short term: due to its tremendous impact, this calls for *broad and systematic* measurement work beyond [46, 48, 97] to gather reliable popularity models.

Second, we find that (ii) simple randomized caching decision and replacement policies perform as well as more complex ones. On the one hand this partly confirms the importance of correlated arrivals, that void the IR assumption. On the other hand, this is a positive finding, as it also suggests that simple policies are able to provide good enough performance in CCN. At the same time we point out that this finding does not include other recent decision schemes worth investigating further [17, 19].

Third, (iii) the impact of the chosen topology is minimal, with respect to the other parameters. This is again not surprising, as [2, 43] suggest request stream aggregation, more than network topology, to be a key factor in cache placement. This also means that,

provided that latency heterogeneity is properly accounted for, the network topology is a less important detail with respect to the previous scenario parameters.

Fourth (iv) this chapter contrasts the performance of NCA under homogeneous vs heterogeneous cache sizes. Specifically, we consider that a given amount of memory resources need to be allocated across a network of arbitrarily connected CCN nodes. We then define several allocation criteria resorting to standard graph centrality metrics – such as betweenness, closeness, stress, graph, eccentricity and degree centralities. Our criteria are either simplistic (i.e., proportional to the centrality metrics) or also incorporate a first degree of technological constraints (i.e., memory quantization in multiple of a given amount).

Although our evaluation is carried out in the context of Content Centric Networks (CCN) it is worth discussing to what extent our findings could apply to other architectures under the Information Centric Networks (ICN) umbrella. We are assisted in this by [1], that overviews similarities and differences between several ICN proposals (namely, DONA, CCN, PSIRP, NetInf), naming, security, routing and forwarding – of which only the latter two are relevant for our purpose. We argue that, in order for our guidelines to apply to an ICN architecture other than CCN, three main assumptions need to hold. First, the ICN architecture must operate as a receiver-driven network of caches. Second, the architecture must partitions objects in chunks. Third, data chunks must backtrack the trail of crumbs left by the request – i.e., data backward path must mirror the request forward path. While the first two assumptions generally hold for any ICN architecture, the latter bares additional discussion. At the same time, even though forward/backward path symmetry behavior is not mandatory for some architectures (e.g., NetInf, Dona), however [1] points out this to be a possible behavior in almost every ICN architecture. Hence, the reach of findings reported in the following may be larger than the narrow CCN scope.

Chapter 7

Forwarding Strategies

In this chapter we focus our attention on the forwarding strategy $\mathcal{F}$, composing a NCA triple. The design space we consider is summarized in Tab. 7.1. The request forwarding strategy $\mathcal{F}$ is constrained by the information available in the forwarding table (FIB). In case CCN nodes have at their disposal FIB information useful to forward requests towards one (or more) server where a persistent copy of the data is stored (e.g., the shortest path to a given content originator), this *information can be exploited* by the CCN architecture. Content can be found only on "en-route caches" (i.e., caches between the content requester and the content originator), possibly failing to find nearby cached copies (e.g., that lie along the shortest path of another close requester). Similarly, data will be cached on the path between the content originator and the requester only.

On the other extreme, *in case no useful FIB information is available, then the neighborhood needs to be explored* to find a temporary copy. In this case, requests are expressed over possibly multiple paths, which can lead to the usual drawbacks of flooding-based algorithms (and require the usual counter-measures, like TTL-based scoping or probabilistic pruning of some branches in the exploration process). Similarly, temporary copies will then be available at multiple neighbor nodes.

Hybrid strategies can be devised when nodes have only partial knowledge of existing routes towards available content items, or in order to exploit nearby temporary replicas. Indeed, given the large mount of content available in the network and the volatility of cached copies, it is not feasible to design a scalable routing protocol able to address all available copies of every item. Accordingly, only part of the requests are deterministically sent over "known" routes, while the others are forwarded via flooding schemes to find "unknown" copies.

Yet another dimension that may draw the exploitation vs exploration design is the *minimal data unit*. At two antipodean extremes, content of interest can be either monolithically requested by users and cached by intermediate routers (*object-level*) or partitioned in chunks (*chunk-level*). Clearly, there is a deterministic overhead when CCN is used in

FIB Knowledge	• None • Omniscient • Partial
Request forwarding strategy	• Exploration • Exploitation • Hybrid
Data unit	• Monolithic object-level • Partitioned chunk-level

Table 7.1: NCA forwarding design space.

chunk-mode, as multiple requests have to be expressed to retrieve the same data. At the same time, requests are generally of limited size, and chunk-mode offers a greater flexibility for data retrieval. Furthermore, requests for the first chunk of any given object could be expressed according to an *exploration* paradigm: this would lead to the discovery of the path toward the closest (cached) object copy, that could be stored in a soft-state FIB cache and *exploited* by requests for subsequent chunks of the same object.

7.1 Exploitation vs Exploration

Based on the previous discussion, in this section we assess pros and cons of the above families of approaches. As our assessment involves both *qualitative* as well as *quantitative* observations, we define and implement a few representative strategies for each family.

Exploitation. Nodes have FIB knowledge concerning the placement of the (possibly multiple) permanent copies of any object in the system. In case multiples original copies are stored in the system, a (uniformly) randomized selection of the server toward which requests are sent is performed. Furthermore, the selection is randomized for each new chunk request in case of chunk-mode CCN architectures. Under the exploitation paradigm, while the number of messages and requests sent is possibly lower, FIB management (i.e., routing, lookup) may have a non-negligible cost, and caching can only happen en-route in the backward path to the originator (so that possibly closest cache is not found with this strategy). Exploitative strategies are the subject of Sec. 7.2.

Exploration. Nodes have no FIB knowledge and are forced to flood requests. Instead of a fixed-scope limit (e.g., by requiring the number of hops n to be $n < TTL_{max}$), we limit the flooding scope probabilistically, such that at the n-th hope, the request message is flooded with exponentially fading probability β^n (as typical in reinforcement learning). Additionally, aggregating requests at nodes, to avoid sending subsequent requests when

a first outstanding request for the same object has not been satisfied yet. As previously introduced, the network is explored only for the first chunk in the case of chunk-mode partitioning. Under exploration, closest copies are expected to be found, at the expense of a more intense communication, that is however possibly limited to the first chunk. Sec. 7.3 is devoted to analyze ideal and realistic exploratory strategies

Hybrid. Nodes may have partial FIB knowledge that allows them to forward requests in an exploitation-based approach, whereas the rest of the catalog will be served by an exploration-based scheme. For instance, since it is likely for popular content to be stored in nearby caches, exploration can be used for the first $\mathcal{K}$th percentile of the request distribution. Instead, for the remaining part of the catalog, the hybrid strategy exploits FIB knowledge (since least popular content it may be necessary to go up to the content originator). Notice that the hybrid case, $\mathcal{K}$ acts as a cutoff parameter, tuning the predominance of exploitation ($\mathcal{K}= 0$) or exploration ($\mathcal{K}= 1$). In this Thesis we do not focus on hybrid strategies, which we widely treat in [10, 11].

7.2 Exploitative strategies

In order to investigate the exploitative strategies, in this section we assume that an external routing process (and name resolution scheme) provides to the strategy layer *multi-path alternatives* from the requester to the content originator. For the sake of simplicity, we limitedly consider at most two alternative paths. The strategy layer has then to decide how to use these paths, on a chunk-by-chunk basis.

We exemplify the situation with a toy-case network in Fig. 7.1, where we consider decisions taken at node A to reach content located in possibly multiple repositories R_1, R_2[1]. On the one hand, (i) in case node A serves requests of user u_1 employing only the shortest-path (single-path, red arrow) to the closest repository R_1 (that pass through B, D), it may miss previous requests of user u_3 (stored at C, E). On the other hand, if strategy layer exploits more than a single path, it can either (ii) employ multiple alternative[2] paths to the same repository (multi-path, green arrow), or (iii) employ multiple shortest paths to different repositories (multi-repository, blue arrow). These *multi-path policies* tradeoff between (i) the fact that shortest path routing cannot fully exploit the benefit of in-network caching, as caching happens only en-route toward the repository, potentially missing closer cached copies; (ii) the fact that alternate repositories may be faraway, so that the interest may travel longer paths; (iii) the fact that path diversity toward the closest repository may be poor, so that few alternative caches are visited beyond those visited by shortest path routing.

[1] In this case $\mathcal{S}(i) \subseteq [D, F], \mathcal{S}(i) \neq \emptyset, \forall i \in \mathcal{N}$ In other words, each content $i \in \mathcal{N}$ can be replicated in one or both repositories R_1, R_2

[2] As primary path, we consider the shortest path between the requester and the originator. As secondary path, we instead use [37] to find the shortest path that is the most diverse from the primary.

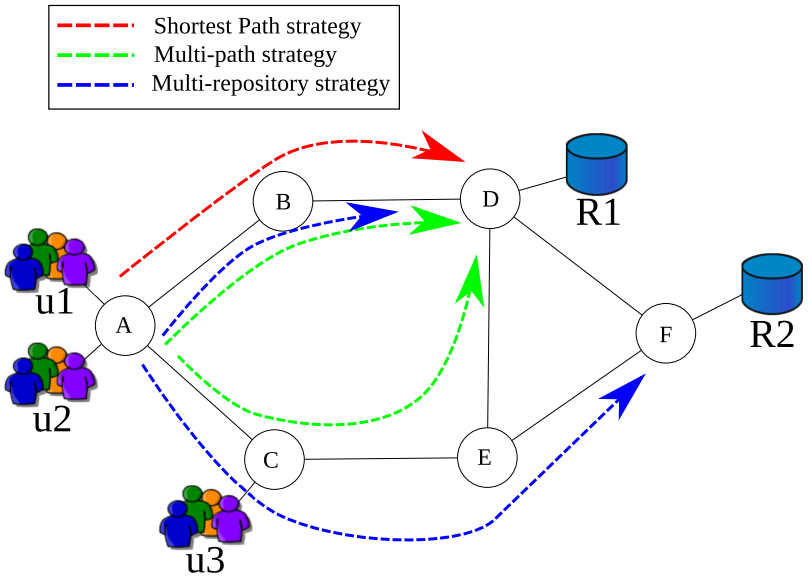


Figure 7.1: Toy-case example.

In both cases, the strategy layer faces further decisions concerning the *multi-path selection* process, such as (i) exploring both paths in parallel, or (ii) alternating between paths (and in this case, if uniformly at random, or deterministically as in round robin). The tradeoff is in this case between between (i) an increased interest load, that violates the flow-balance and possibly causes cache contention and (ii) a possibly slower convergence to the repository.

Finally, the selection process can happen either (i) every chunk or (ii) once per object (keeping thus a per-object state about the path). We denote this choice as *multi-path retention*. In case (i), decisions are taken for every chunk, so no path is retained. Conversely, in case (ii) an interest[3] is sent over all the possible paths at time 0, and the path retained is the one along which the first data will come. The path retention tradeoffs (i) simplicity for (ii) greater efficiency, as it may be preferable to forward interest *toward the closest cached copy* (as opposite to the closest repository). As we have seen that caching policies and topologies play a limited role in determining system performance, we now fix the ⟨LCE,LRU⟩ caching policies and limitedly consider the Geant network to assess the impact of the *strategy layer*. In this case we use $\alpha = 1$ to consider a more realistic scenario. We further consider the case where all links have homogeneous propagation delay, and the heterogeneous case with real delays reflecting the interconnection lengths.

[3]In case parallel path are used, the same interest is expressed over multiple paths; otherwise, a different chunk is requested over each alternative path.

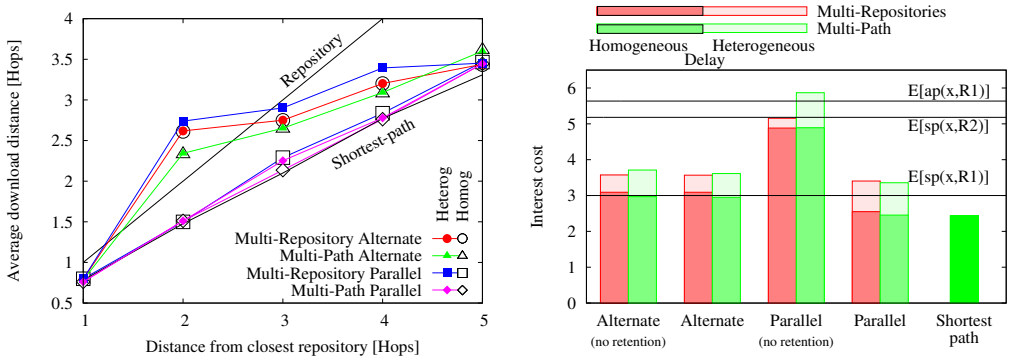


Figure 7.2: Impact of strategy layer on average distance from a cache, and interest/data cost.

To promote cross comparison we resort to Geant publicly available network topology (see Tab. 5.1). Overall, we tested 12 multi-path strategy layers, and we report only the most interesting combinations in the following.

7.2.1 Performance evaluation

Fig. 7.2(a) reports the average download distance as a function of the node distance from the closest repository. Notice that filled points refer to scenario with heterogeneous delay, empty ones to homogeneous delay. Two lines report reference for cases where the content is entirely downloaded from the closest repository, or is downloaded with shortest path routing. Fig. 7.2(b) reports instead the average interest load, i.e., the number of links that each interest has crossed before hitting a cache. Light-gray color is used for heterogeneous delay, dark-gray for homogeneous delay. Three lines report the average distance from the closest repository $E[sp(x, R1)]$, the shortest path distance from the alternate repository $E[sp(x, R2)]$, the length of the alternate path toward the closest repository $E[ap(x, R1)]$.

As expected, when multiple paths are used, these are longer than the distance to the closer repository, hence the download distance increases. Looking at Fig. 7.2(a), this effect is especially noticeable only for nodes that are close to the repository, with a peak at two hops; conversely, faraway nodes will likely find the content cached somewhere before the repository, with performance closer to shortest path routing. Similarly, when multiple paths are used in parallel, the interest load roughly doubles. Considering Fig. 7.2(b), path retention may however successfully mitigate the interest load increase, at a price of keeping per-object state in CCN routers.

We point out that in case paths are used in an *alternate* fashion, there is no difference between a uniform randomized criterion and a deterministic round-robin one (hence, we do not show this in the plots). Intuitively, if deterministic decisions are not coordinated be-

tween nodes, they are equivalent to random decisions from the overall network perspective. When instead multiple paths are used in a *parallel* fashion, this lead to a more aggressive probing policy, that has higher chances to find closer cached content.

Also, we see that multiple alternative paths *toward the same repository* should be preferred to the use of multiple shortest paths toward several repositories (furthermore this lead to more robust choices in case of heterogeneous delay). Intuitively, this is because CCN values diverse paths as they explore different caches, more than because they ultimately lead to different repositories. Interestingly, this happens despite the average alternate path toward the closest repository is on average longer than the shortest path toward the alternate repository, i.e., $E[ap(x, R1)] > E[sp(x, R2)]$ in Fig. 7.2(b).

The above observation has an important consequence. First, this means that, unless the ultimate aim is to reduce load at the repositories, information concerning a single repository suffices. Furthermore, in CCN efficiency is maximum when the closest cached copy is found: since cache update dynamics are much faster than routing dynamics possibly, this information will likely not be available in the FIB. Hence, there may no need for the routing protocol to disseminate information about optimal alternate paths to the repository either, since an opportunistic exploration of a CCN node neighborhood may suffice.

Overall, we find that naïve multi-path strategies exploiting other paths than the shortest one may lead to explore a longer distance – increasing the overall network load, and reducing the cache hit rates. At the same time, we recognize that multi-path benefits come from an increased resilience (in case shortest path fail) and to a possibly reduced load at the repository (whose uplink bandwidth can be a scarce resource with respect to router capacities). In this case, it seems that multi-path hybrid strategies that complement single-path routing with an opportunistic exploration of a CCN node neighborhood may be worth exploring. In this case, indeed, we avoid the issue of distance increase, letting the interest find the nearest copy.

7.3 Exploratory strategies: toward iNRR

In this section, we leave exploitation approaches, on the behalf of exploration techniques. As said above, exploitative strategies tend to averagely increase (compared to the shortest path) the distance traveled by data content. This is because data is constrained on predetermined paths. Instead, an ideal name-based routing protocol would have to address all temporary copies of every content item in order to forward user requests towards the "best" available replica (e.g., closest). We name this ideal strategy iNRR(ideal Nearest Replica Routing). iNRR is clearly unfeasible in a realistic CCN network for different reasons:

- The *network scale*; CCN paradigm applies to content of different applications and is not intended to be confined to small, controlled network regions.
- The *time scale*; temporary copies stored at network nodes are highly volatile and the

signaling overhead involved by frequent route updates would be excessive.

- The *forwarding table (FIB) size*, that is already a matter of concern within all CCN designs, even considering only *permanent* content copies rather that network-cached temporary replicas [102].

Thus, we tackle the problem of the definition, analysis and implementation of a scalable forwarding policy $\mathcal{F}$, suitable for CCN, which however approaches toward the iNRR forwarding. Our goal is to design a forwarding strategy that: (i) can discover temporary content replicas and forward requests accordingly; (ii) requires little or no a priori knowledge; (iii) does not generate too much additional signaling overhead; (iv) can achieve implicit cache coordination.

7.3.1 Coupling forwarding and exploration

Given the pervasiveness of caches in CCN, meta-caching is considered a crucial element of a NCA to differentiate content of individual caches. Forwarding is instead essential to extend the reach beyond caches that lay on the path toward the repository, possibly reaching off path copies. Yet, while NCA performance are dependent on the triple $\langle \mathcal{F}, \mathcal{D}, \mathcal{R} \rangle$, with few exceptions (see Sec. 7.2), research has so far limitedly considered a single of the above aspect in isolation. More specifically, work focusing on $\mathcal{F}$ aims at implementing policies to efficiently reach off-path caches, and assumes that newly arriving content is always cached [9–14], [3–8] – generally referred to as Leave a Copy Everywhere (LCE) in meta-caching terms. Work focusing on $\mathcal{D}$ instead aims at implementing policies to reduce cache redundancy, and assumes that requests are forwarded according to Shortest Path Routing (SPR) [15–19]. Most importantly, a debate has been recently ignited around the usefulness of ubiquitous caching [2, 103].

In particular, very recent work [2] shows that the most of the caching gain is attainable by simply (and painlessly) caching at the edge of the network. Yet, we argue that [2] misses a crucial point: i.e., that the interaction of the above policies concurs in determining the global NCA performance. While authors of [2] correctly select an ideal forwarding policy $\mathcal{F}$, that achieves optimal forwarding decisions, their (implicit) selection of the $\langle \mathcal{D}, \mathcal{R} \rangle$ pair (and especially of the meta-caching policy $\mathcal{D}$ that, as we will see, plays a paramount role) yields to significant underestimation of the achievable NCA performance. Tab. 7.2 reports a taxonomy of related work. The table is split in two portions, meta-caching (top) and forwarding (bottom): it clearly emerges that $\mathcal{F}$ and $\mathcal{D}$ aspects have been so far studied separately. Work focusing on meta-caching usually assumes Shortest Path Routing (SPR) as underlying request forwarding strategy. In this context, many policies have been proposed that are either deterministic (LCE, LCD [15, 16], Betweenness [19]) or probabilistic (Fix [15, 44], ProbCache [17], WAVE [18]). These policies exploit different information (ranging from simple distance [15, 17] to more complex topological properties [19]) and possibly explicitly take into account CCN chunking [18].

Similarly, work focusing on forwarding policies usually assumes that new contents are always cached, which is commonly referred to as Leave a Copy Everywhere (LCE) in meta-caching terms. The interest of alternative strategies to SPR is that there may be closer cached copies laying *off path* between the requester and the custodian of the permanent copy, that thus SPR is unable to reach. To achieve this purpose, the ICN community has tested several forwarding approaches, ranging from multiple disjoint source routed paths [9], to dynamic approaches based on flooding [10], learning [11, 13], or routing using potential [12]. Of particular interest, [2] considers an ideal Nearest Routing Replica (iNRR) scheme that allows to reach the closest, possibly off path, cached copy. While iNRR is not a practical scheme, as it requires instantaneous knowledge of the status of all caches in the network, however it provides an ideal upper-bound to $\mathcal{F}$ performance, and as such is worth considering. Additionally, we offer two distributed NRR implementations in Sec. 7.3.4, that can attain performance arbitrarily close to that of iNRR.

Modeling. Concerning modeling work, separation of $\mathcal{F}$, $\mathcal{D}$ and $\mathcal{R}$ is easily understood: due to the complexity in analysing caching networks, studies have tackled each aspect in isolation. In particular, considering simple topologies (e.g., cascades or trees), [16] models LCD meta-caching ($\mathcal{D}$ policy), while [6] addresses LRU and random replacement ($\mathcal{R}$ policies), and [3, 4] explicitly account for the fact that objects are split in chunks. Considering instead more complex networks, [8] models object-level cache hit of Shortest Path Routing (SPR) on arbitrary topologies.

We point out that despite CCN introduces a number of new challenges (e.g., chunk vs object level, pervasive caching, request routing over complex topologies, etc.), caching is not a new problem. As such, in terms of modeling techniques, the above work possibly extends to the NCA context previous seminal work. More precisely, [6, 16] build over the Che [7] approximation, while the MMPP model in [3, 4] extends Jelenkovic [5] to the case of multiple chunk and [8] extends the Dan and Towsley [104] LRU approximation from a single cache to a network of caches. In Sec. 7.3.3 we extend [8] by considering alternative forwarding policies to SPR, and in particular iNRR (in reason of its performance as we will see shortly).

Simulation. Separation of $\mathcal{F}$, $\mathcal{D}$ and $\mathcal{R}$ is instead less justified in simulative work. In part, this is due to the fact that concerning $\mathcal{R}$ a natural choice is the Latest Recently Used (LRU) replacement, though it has been pointed out that random replacement (i) exhibits similar performance at a lower complexity [6, 9, 45] (ii) it may be preferable to LRU due to line rate constraint [44, 47]. We further point out that, while the joint impact of meta-caching $\mathcal{D}$ and replacement $\mathcal{R}$ policies has gained limited attention (among others, by our own work [9]), to the best of our knowledge, the forwarding $\mathcal{F}$ and meta-caching $\mathcal{D}$ policies have not been jointly considered so far. As the performance impact of the $\langle \cdot, \mathcal{D}, \mathcal{R} \rangle$ couplet is limited with respect to that of $\langle \mathcal{F}, \mathcal{D}, \cdot \rangle$, in this work we mostly focus on the latter. We start by showing this impact in Sec. 7.3.2. Then, we critically contrast recent results that

Table 7.2: Meta caching and forwarding strategies.

Meta-caching $\mathcal{D}$	**Type**	**Knob**	**Ref**
LCE	Deterministic	-	[2–19]
Fix	Probabilistic	p	[9, 15, 44]
ProbCache	Probabilistic	Distance	[17]
LCD	Deterministic	Distance	[9, 15, 16]
WAVE	Probabilistic	Distance	[18]
Btw	Deterministic	Centrality	[19]

Forwarding $\mathcal{F}$	**Type**	**Knob**	**Ref**
SPR	Deterministic	-	[2–19]
Source routing	Probabilistic	-	[9]
Flooding	Probabilistic	*Distance*	[10]
INFORM	Probabilistic	*Delay*	[11]
CATT	Deterministic	*Distance*	[12]
NDN	Deterministic	*Dist/Delay*	[13]
iNRR	Deterministic	-	[2]

(too) quickly dismiss ubiquitous caching [2].

7.3.2 Performance evaluation

We start by showing that, provided that forwarding and meta-caching decisions are jointly considered, gain of ubiquitous caching can be quite significant. Simulation results are obtained with ccn$\mathcal{S}$im [105], an highly scalable chunk-level[4] simulator that we have developed and optimized over the last few years. To give an idea of ccn$\mathcal{S}$im scalability, the large-scale scenario reported in Sec. 6.1, corresponding to one billion worth of object requests, out of a 100 million object catalog, with caches storing 100,000 objects, can be simulated by a common off-the-shelf PC equipped with 8GB of RAM memory in few hours [105]. For this work, we extended ccn$\mathcal{S}$im to include a number of meta-caching[5] (e.g., ProbCache [17], Btw [19]) and forwarding (e.g., iNRR [2]) algorithms, that we will soon make available in a new ccn$\mathcal{S}$im release [101].

[4] To facilitate comparison with [2, 8] that consider object-level caching, in this work we use ccn$\mathcal{S}$im at object level.

[5] As we do consider object-level caching, we point out it does not make sense to include WAVE [18] in the comparison.

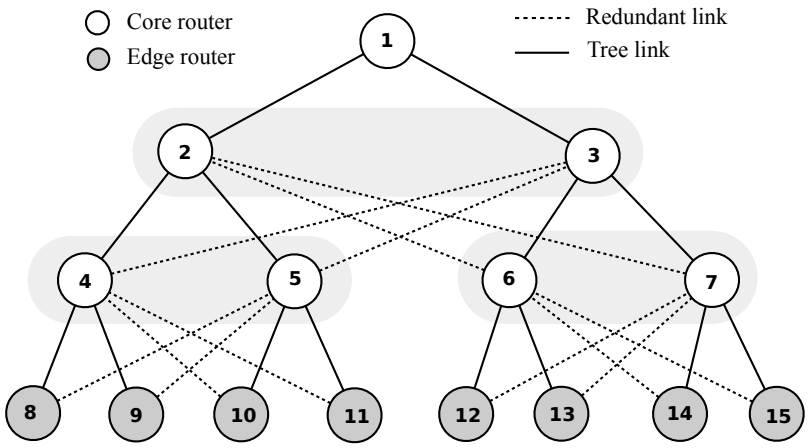


Figure 7.3: Redundant 4-level binary tree. Dashed links are present with probability μ. Shadowed blocks represent aggregate caches seen by lower level nodes in presence of redundancy ($\mu > 0$).

7.3.2.1 Scenarios

To facilitate comparison with [2] in this section and with [8] in Sec. 7.3.3, we consider network scenarios as similar as possible to those introduced there, namely grid [8] and access tree topologies [2]. We point out that [2] additionally considers access trees to be attached to PoP of realistic backbone networks (gathered with Rocketfuel as in our previous work [9]). Despite great effort is made in [2] to describe the scenario, however the lack of crucial parameters (e.g., repository placement, content redundancy and allocation to repositories, etc.), makes a 1-to-1 comparison difficult. As such, to promote cross-comparison, we make our scenarios available to the scientific community, under the form of configuration files for ccn$\mathcal{S}$im, so that independent research can confirm (or disprove) our findings.

Specifically, we consider a 10x10 grid (100 nodes) and a 6-level binary tree ($2^6 - 1$=63 nodes). Since networks are engineered adhering to fault tolerance and resilience principles, it is extremely unlikely for an access topology to have exactly a single link between any pair of parent and child nodes as in [2] – as otherwise, cutting a single link up in the hierarchy would cut a whole subtree. As such, we consider that a node may have an additional link to its aunt (i.e., the immediate sibling of its direct parent) that can be used for backup or load balancing. We model the presence of these additional links (represented with dashed lines in the 4-level tree of Fig. 7.3) by the means of i.i.d. probability $\mu \in [0, 1]$.

For simplicity, we consider topologies with uniform delay (1ms), as heterogeneity plays a minor role [9,106], and consider to operate below congestion (links have infinity capacity).

As in [2], that offers fitting over global Akamai dataset[6], we consider object popularity to follow a Zipf distribution with $\alpha \approx 1$. We use a cache to catalog size ratio of 0.1% (much more conservative that 5% in [2]) instantiated in a small (large) scenario where caches are able to store 100 (100,000) objects out of a 100,000 (100,000,0000) objects catalog. Small to large scale scenarios allow us to respectively explore wide parameter settings, and gather performance on a more realistic use case. We adopt the following methodology. Simulations start from empty caches, but statistics are gathered after that caches are filled, and the hit ratio converges: in other words, we consider the system in steady state. Results are averaged over 20 runs.

7.3.2.2 Performance

As performance metric, we consider the average distance that the content has traveled in the CCN network. This metric has the advantage of being very insightful and compact at the same time: it relates to user QoE (i.e., delay) as well as network QoS (i.e., load and cache hit). Moreover, while [2] additionally expresses cache hit and repository load (see Sec. 6.1.3), it however mostly reports relative error between iNRR and alternate strategies: as direct comparisons are de facto impossible, and to limit redundancy given space constraints, we hence avoid reporting additional metrics beyond the content distance.

In terms of $\mathcal{F}$, instead of being limited by implementation (and configuration) details of the numerous NCA forwarding policies proposed [9–13], we consider (i) iNRR [2] as upper-bound of the achievable performance for off-path caching, and (ii) SPR which we expect to correspond to the on-path caching lower-bound. In terms of meta-caching $\mathcal{D}$, we instead implement several of the proposals in Tab. 7.2. We include LCE as a term of comparison, that we again expect to provide a performance lower-bound as it provides poor cache diversity and forces high eviction rates over the whole network. Finally, in terms of replacement $\mathcal{R}$ we experiment with LRU and uniform probabilistic replacement [44, 45] (though we mostly report results concerning the former due to secondary $\mathcal{R}$ impact).

Fig. 7.4 reports the average distance at which content is found in the CCN network as function of the meta-caching policies, for SPR (left) and iNRR (right) forwarding, on tree (top, without additional links $\mu = 0$) and grid topologies (bottom). The plot is annotated with percentage gain that could be achieved by moving from the ⟨SPR,LCE⟩ worst case to other, more sensible, ICN configurations.

First, recall that the top plots of Fig. 7.4 report the scenario of [2]: in this case, [2] correctly points out that the difference between ⟨SPR,LCE⟩ and ⟨iNRR,LCE⟩ is below 10%. Yet, as authors limitedly experiment with a naive LCE meta-caching, they ignore potential gain due to LCD (about 21% even considering SPR) or the joint use of LCD and iNRR (about 26%). Additionally, we point out that gains in this scenario are limited by the poor path diversity that the tree offers to iNRR. For instance, in the example provided

[6] Actually, Fig1 and Tab1 of [2] are in disagreement, since the number of objects is larger than the number of requests. Hence, "Request" in Tab1 of [2] is a misspell for "Objects".

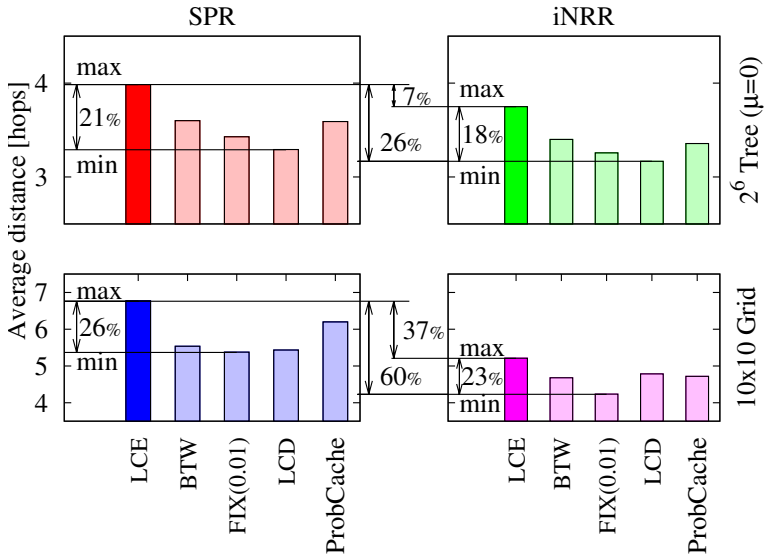


Figure 7.4: ⟨F,D⟩ performance at a glance: average content distance as a function of meta-caching policies, for SPR (left) and iNRR (right) forwarding, on tree (top) and grid (bottom) topologies.

in Fig. 7.3 for a 4-level tree, starting at node 15, on-path caching with SPR traverses 4 caches from the edge to the root (i.e., 15, 7, 3, 1): iNRR disposes of only 2 additional off-path nodes when $\mu = 0$ (i.e., 6, 14), but of 6 nodes when $\mu = 1$ (i.e., 4, 5, 6, 12, 13, 14).

Hence, gains of ⟨iNRR,LCD⟩ are potentially higher on more meshed topologies, that allow iNRR to explore a larger (and closer) neighborhood. This clearly reflects in the bottom plot of Fig. 7.4, obtained on a 10x10 grid: in this case, the difference between ⟨SPR,LCE⟩ and ⟨iNRR,LCE⟩ is about 37%. Additional gain could be attained by coupling iNRR forwarding to LCD or fixed probabilistic FIX decisions, for a reduction of the average distance of about 60%. Clearly, as content travels less than half the path in ⟨SPR,LCE⟩, the network load also divides by over a factor of two, and similarly happens for user latency – shortly, opposite to findings in [2], there seems to be a case for ubiquitous caching after all.

7.3.2.3 Sensitivity analysis

Fig. 7.5 reports a sensitivity analysis of the meta-caching policies, gathered via simulation over smoothly varying network redundancy $\mu \in [0, 1]$. The plot is annotated with gain from ⟨SPR,·⟩ to ⟨iNRR,·⟩, as well as with gain due to the redundancy (from $\mu = 0$ to $\mu = 1$ for any given $\langle \mathcal{F}, \mathcal{D} \rangle$ setting).

As it can be expected, redundancy plays a negligible role for SPR (though in case of

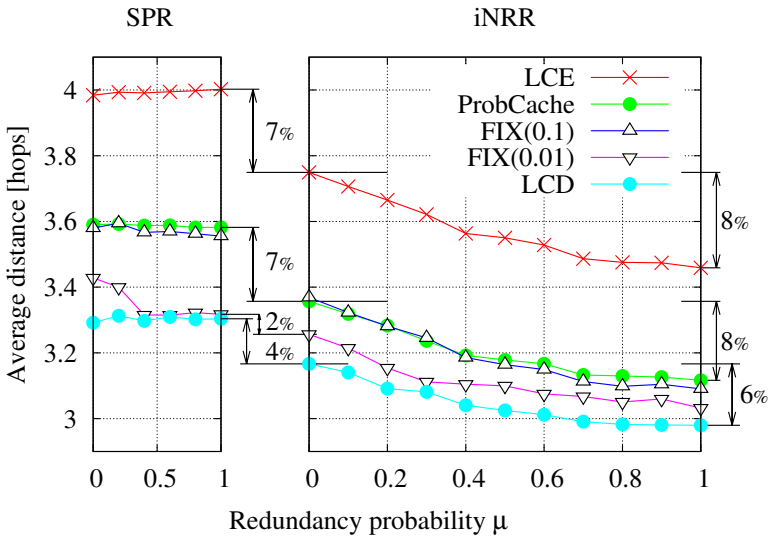


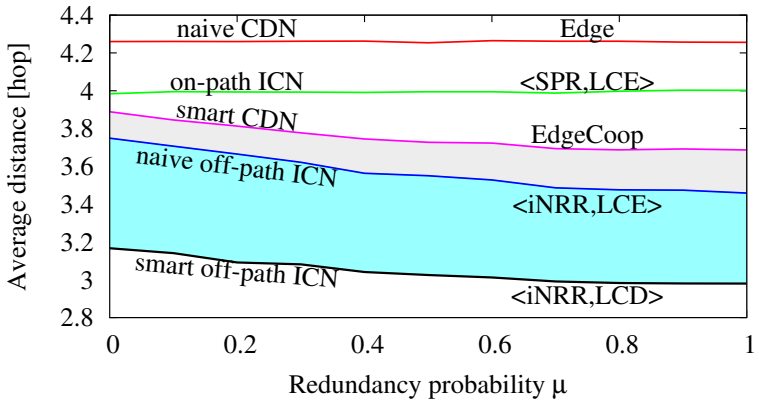
Figure 7.5: Sensitivity analysis of $\langle \mathcal{F}, \mathcal{D} \rangle$ when $\mathcal{R}$=LRU: 6-level binary tree topology, with varying redundancy probability μ.

multiple equivalent paths, SPR chooses between them at random, possibly traversing different caches). Unsurprisingly, deterministic LCD decisions consistently achieve best performance for trees [15], exhibiting furthermore a good interplay with iNRR. Next comes simple probabilistic decisions FIX($\frac{1}{100}$), while complex probabilistic strategies driven on either distance (ProbCache [17]) or topological properties (e.g., Btw [19]) achieve intermediate gain. In reason of the added complexity (as it is often pointed out, simpler solutions are preferable due to line rate constraints [44, 47]) and limited gain, we thus disregard the latter meta-caching policies, while we point out simple probabilistic decisions to be a good enough candidate for ICN.

Overall, it can be seen that average path length increases from slightly less than 3 hops for ⟨iNRR,LCD⟩ to about 4 hops for ⟨SPR,LCE⟩, i.e. a sizeable 33% increase (though gain may be larger for more meshed topologies). Finally, we experiment with different Zipf skew settings: while we do not report pictures for reason of space, we observe that gain increases for growing α.

7.3.2.4 Comparison with edge-caching

We perform an exhaustive comparison with some of the edge-caching techniques (namely, Edge, EdgeCoop) that [2] offers as "good enough" replacement for ICN/CCN. We again focus on the access tree topology, to mimic scenario in [2], and additionally consider that networks are possibly engineered with fault tolerance (i.e., redundancy probability μ).



μ	$\langle SPR, LCE\rangle$	Edge Coop	$\langle iNRR, LCE\rangle$	$\langle iNRR, LCD\rangle$
0	2%	5%	9%	28%
1	2%	11%	18%	37%

Figure 7.6: Comparison with Edge and EdgeCoop caching strategies [2]: average distance $E[d_X]$ of strategy X, as a function of the redundancy probability μ. Additionally, the figure tabulates the gain of strategy X over Edge, measured as $(E[d_{\text{Edge}}] - E[d_X])/E[d_{\text{Edge}}]$.

We consider the Edge strategy where only leaf nodes have caching space, corresponding to a Content Distribution Network (CDN) scenario. We next consider ⟨SPR,LCE⟩ where caching is ubiquitous but, due to SPR forwarding strategy, only on-path caches can be exploited. We further include ⟨iNRR,LCE⟩ that [2] (wrongly) considers to achieve best ICN performance. We then implement EdgeCoop where "CDN routes can do a scoped lookup in sibling cache" [2]. As the terse EdgeCoop description in [2] does not allow to understand whether only caches having a common parent can cooperate, we opt for an approach that is as favorable as possible to EdgeCoop, to avoid any bias toward ICN. Our implementation of EdgeCoop allows caching only at leaf nodes, but exploits iNRR routing strategy: thus, any temporary copy that is cached at a distance shorter than or equal to that of the permanent copy stored at the custodian above the root of the tree is possibly accessed (in practice, only half of the leaf nodes are accessible when $\mu = 0$, while all leafs are accessible for $\mu = 1$). Finally, we include the ⟨iNRR,LCD⟩ configuration, representing the (true) best baseline for CCN.

Two shaded regions are shown in the plot. On the basis of the light-gray region separating EdgeCoop from ⟨iNRR,LCE⟩, authors in [2] conclude that ICN does offer only minimal performance improvement over sensibly configured CDN scenarios. The dark-gray region between ⟨iNRR,LCE⟩ and ⟨iNRR,LCD⟩ instead represents the potential gain due to joint meta-caching and forwarding that other work, including [2], has

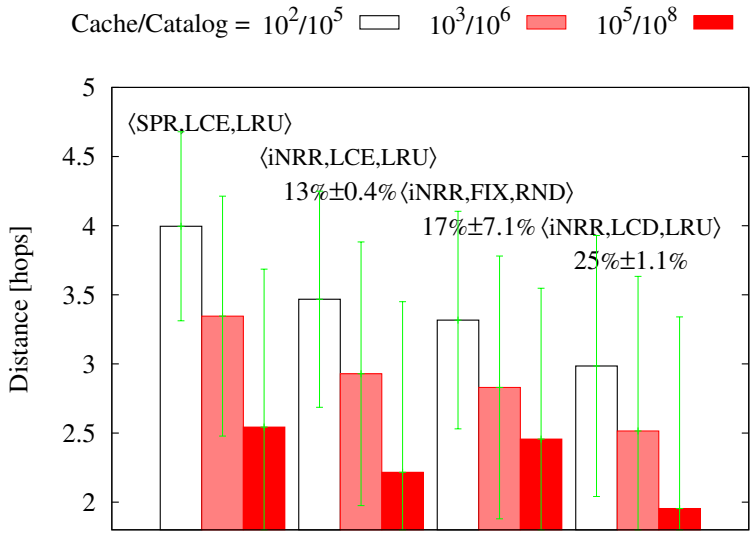


Figure 7.7: Small- vs medium- and large-scale scenarios. Relative gain of over ⟨SPR,LCE,LRU⟩ remains similar over all scenarios.

missed so far. Finally, the figure tabulates gain of a strategy X over Edge, computed as $(E[d_{\text{Edge}}] - E[d_X])/E[d_{\text{Edge}}]$ where $E[d_X]$ represents the average distance of strategy X. It can be seen that ⟨iNRR,LCD⟩/Edge gain is significantly higher than EdgeCoop/Edge gain, for both binary trees ($\mu = 0$) and trees with full redundancy ($\mu = 1$).

7.3.2.5 Small to large-scale scenarios

Small, medium (or large) scale scenarios allow us to respectively explore wide parameter settings, and gather performance on a more realistic (or extreme) use case. We fix $\alpha = 1$ and the cache to catalog size ratio C/N to a conservative 0.1%, and let the cache C and catalog sizes N vary. Precisely, we instantiate a small-scale scenario with $C/N = 10^2/10^5$, a medium-scale with $C/N = 10^3/10^6$ and a large-scale with $C/N = 10^5/10^8$. As video is preeminent, and given an average size of YouTube videos of 10MB [46], medium and large scale cache sizes vary in the feasible 10GB [44, 47] to challenging 10TB [107] range. Catalog size of the medium scenario is of the same order of magnitude of [2], whereas the large-scale scenario models a more challenging YouTube scenario [48].

Average distances are reported in Fig. 7.7 (along with the coefficient of variation) for naive on-path caching ⟨SPR,LCE,LRU⟩, naive off-path caching ⟨iNRR,LCE,LRU⟩, simple probabilistic off-path meta-caching and replacement ⟨SPR,FIX,RND⟩, and the best off-path strategy ⟨SPR,LCD,LRU⟩. Each strategy is annotated with the average gain over ⟨SPR,LCE,LRU⟩ (± standard deviation across different scales).

We see that performance improve (i.e., distance decreases) for large catalogs. This can be explained considering that, for fixed Zipf $\alpha = 1$ and fixed cache to catalog ratio C/N, a larger cache C can accommodate a larger fraction of top content out of the entire catalog N. Formally, $\sum_i^C i^{-\alpha} / \sum_i^N i^{-\alpha}$ increases from small to large catalog, so that C=100 (100,000) most popular cached objects corresponds to the 43% (63%) of the whole requests for a N=100,000 (100,000,000) catalog.

Hence, we gather that small scale scenario (i) corresponds to conservative cache hit results and (ii) allows a reliable estimate of the relative gain of ubiquitous caching over on-path caching – as the relative gain over ⟨SPR,LCE,LRU⟩ is the same for all scenarios (except the simplistic ⟨SPR,FIX,RND⟩ case we disregard in the following).

7.3.3 Modeling iNRR

As shown in the previous section, iNRR achieves interesting performance with respect to SPR forwarding. Furthermore, iNRR benefits are especially apparent with topologies having redundant links. As such, it would be useful to have an approximate iNRR model valid for arbitrary network of caches. We tackle this challenge by extending the aNET model proposed in [8], that unlike other caching models is applicable to any topology [8] but limitedly focuses on Shortest Path Routing.

7.3.3.1 aNET model and notation

According to our terminology, aNET [8] models a ⟨SPR,LCE,LRU⟩ network. aNET approximates network behavior by decomposing the problem and computing the LRU approximation [104] for each cache in the network. Network is represented as a graph $G = (V, E)$ with $v \in V$ a vertex node having a cache of size $|v|$ objects. We denote the content catalog with $\mathcal{N}$, with size $N = |\mathcal{N}|$. As ⟨SPR,LCE,LRU⟩ forwards the miss stream of each cache along the SPR toward the permanent replica, it follows that the incoming request stream at each cache accounts for both exogenous user request, as well as the miss stream of neighboring caches. aNET takes into account this incoming stream by iterating the solution of individual caches, and reevaluating the miss stream until the stabilization of the whole system. aNET iteratively solves the following set of equations:

$$r_{i,v} = \lambda_{i,v} + \sum_{u:R(u,\mathcal{S}(i))=v} m_{i,u} \tag{7.1}$$

$$p_{i,v} = \frac{r_{i,v}}{\sum_{j=1}^{N} r_{jv}} \tag{7.2}$$

$$\vec{\pi}_v = LRU(\vec{p}_v, |v|) \tag{7.3}$$

$$m_{i,v} = r_{i,v}(1 - \pi_{i,v}) \tag{7.4}$$

Incoming requests at node v for content $i \in \mathcal{N}$ are expressed in Eq. (7.1). The first term in Eq. (7.1) represents the exogenous arrival rate $\lambda_{i,v}$ for content i. The second term of

Eq. (7.1) accounts for the miss stream $m_{i,u}$ coming from neighboring nodes u having v as their next hop $R(u, \mathcal{S}(i))$ in the shortest path toward the repository $\mathcal{S}(i)$ for content $i \in \mathcal{N}$. The local popularity $p_{i,v}$ is expressed by Eq. (7.2), representing the relative proportion of request of content i at node v. Given the steady state local request distribution over all contents $\vec{p}_v$ and a cache size $|v|$, each cache v applies in Eq. (7.3) the LRU algorithm [104] to determine the probability $\vec{\pi}_v$ that any given content $i \in \mathcal{N}$ is present in the cache of v at any given time. Finally, the miss stream $m_{i,v}$ is computed as in Eq. (7.4).

Two crucial points in the above set of equations are worth stressing. First, Eq. (7.4) was only proven to hold for an Independent Reference Model (IRM) [8]. Second, the approximate LRU algorithm Eq. (7.3) was designed only for IRM streams [104]. However, as the request stream also consists of miss stream of the neighbors as per Eq. (7.1), the aggregate request stream is not IRM: hence, steps Eq. (7.3)-Eq. (7.4) consist in an IRM violation, and are potential sources of error in the approximation.

7.3.3.2 iNRR model

We extend the set of aNET equation to model iNRR forwarding strategy. Under SPR forwarding, content can be possibly found only along the shortest path toward a custodian of permanent content replicas: hence, the miss stream Eq. (7.1) aggregates requests of shortest paths passing through v. The crucial difference from aNET is that, under iNRR forwarding, any *valid* path is possibly followed. By valid path, we imply that (i) paths are loop free, (ii) in case multiple copies are stored at several nodes along any given path, the closest copy is accessed. Additionally (iii) in case of multiple copies having equal distance over multiple paths, each copy is equally likely to be chosen.

To model the above observations (i)–(iii), we introduce the following notation. As in aNET, the SPR routing matrix for the network $R(v, u), v, u \in V$ indicates v's next hop to reach node u. Nodes are directly connected to v when $R(u, v) = v$, and we indicate with $N(v) = \{u : R(u, v) = v\}$ the set of v's neighbors. For convenience, $\mathcal{S} = \mathcal{S}(i)$, $\forall i \in \mathcal{N}$ indicates the unique repository in the network (the model can be easily extended to the case of multiple repositories), so that $R(v, \mathcal{S})$ represents the FIB information used by SPR to reach the custodian.

In addition to SPR FIB information (possibly hitting content cached on-path to $\mathcal{S}$ as in aNET), iNRR is able to find any off-path content that is not located further than the repository (so that caches as close as the repository, can offload the latter). To identify such content, we define $D(v, u)$ as the SPR distance between any two nodes $v, u \in V$. We next define $B(v, u)$ as the ball centered in v having ray $D(v, u)$, i.e., $B(v, u) = \{x \in V : D(v, x) \leq D(v, u)\}$. Thus, $B(v, u)$ represents the set of nodes that are not further away than u from v. For convenience, we also define the border and interior of $B(v, u)$ as $B_b(v, u) = \{x \in V : D(v, x) = D(v, u)\}$ and $B_i(v, u) = \{x \in V : D(v, x) < D(v, u)\}$ respectively. For instance, $B_b(v, \mathcal{S})$ represents the set of nodes that are as far from v as the server $\mathcal{S}$, while $B_i(v, u)$ represents the set of nodes closer than u to v. Finally, we denote

with $m_{i,u,v}$ the proportion of miss stream for content i coming from u to v. Then, our iNRR model iteratively solves the following set of equations $\forall i \in \mathcal{N}, v \in V$:

$$r_{i,v} = \lambda_{i,v} + \sum_{u:u\in N(v)} m_{i,u,v} \tag{7.5}$$

$$p_{i,v} = \frac{r_{i,v}}{\sum_{j=1}^{N} r_{jv}} \tag{7.6}$$

$$\vec{\pi}_v = LRU(\vec{p}_v, |v|) \tag{7.7}$$

$$m_{i,v} = r_{i,v}(1 - \pi_{i,v}) \tag{7.8}$$

$$s_{i,v,u} = \sum_{\substack{x:R(v,x)=u \\ \wedge\, x\in B(v,\mathcal{S})}} \left[\prod_{y\in B_i(v,x)} (1-\pi_{i,y})\right] \frac{\pi_{i,x}^2}{\sum_{z\in B_b(v,x)} \pi_{i,z}} \tag{7.9}$$

$$m_{i,v,u} = \begin{cases} m_{i,v} s_{i,v,u} & u \neq R(v,\mathcal{S}) \\ m_{i,v}(1 - \sum_{w\neq u} s_{i,v,w}) & u = R(v,\mathcal{S}) \end{cases} \tag{7.10}$$

Shortly, while Eq. (7.6), Eq. (7.7) and Eq. (7.8) perform the same steps as in aNET, iNRR modifies Eq. (7.5) to account for a proportion of miss stream of neighboring nodes, and further adds equations Eq. (7.9) and Eq. (7.10) to precisely quantify this proportion.

As per observation (iii), any node u will split its miss stream equally among its neighbors $N(u)$. This is modeled by Eq. (7.5), where all v's neighbors $N(v)$ contribute to request arrival at v, with $m_{i,u,v}$ the proportion of miss stream for content i coming from u. Observations (i) and (ii) are instead expressed through Eq. (7.9) and Eq. (7.10). More precisely, Eq. (7.9) defines the split ratio $s_{i,v,u}$ among neighboring nodes, and Eq. (7.10) applies the split ratio to the miss stream $m_{i,v}$, depending on whether u lays on the shortest path to the server $u = R(v,\mathcal{S})$ or not.

Especially, Eq. (7.9) bares additional discussion. The term $s_{i,v,u}$ represents the proportion of the miss stream of node v sent through v's immediate neighbor u to reach node x for content i. The conditions upon which iNRR forwards such requests are:

- Next hop for x from v passes through $u = R(v,x)$, and the distance $D(v,x)$ is shorter than or equal to the distance toward the server $R(v,\mathcal{S})$, i.e., x falls in the ball $B(v,\mathcal{S})$ (external sum).
- Any node y closer than x to v, i.e., laying in the interior ball $B_i(v,x)$, does not own the content i, which happens with probability $1 - \pi_{i,y}$ for each node (internal product).
- The selected node x owns the item i (with probability $\pi_{i,x}$), and it is chosen among all the nodes $z \in B_b(v,x)$ at the same distance from v (terms $\pi_{i,x}/\sum_z \pi_{i,z}$).

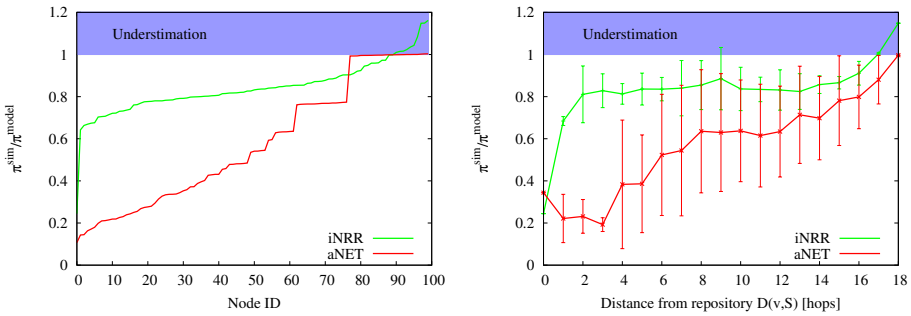


Figure 7.8: iNRR vs aNET hit rate π accuracy: per-node (top) and as function of the distance from the repository (bottom). Grid 10x10 topology.

Finally, by means of Eq. (7.10), we differentiate the case in which the neighbor $u = R(v, \mathcal{S})$ is the immediate next hop toward the repository or not, giving preference to cached copies to offload the repository. Hence, the miss stream that finds objects in the ball $B(v, \mathcal{S})$ flows through off-path neighbors, whereas the rest of the miss stream flows through the next hop $u = R(v, \mathcal{S})$, thus on-path to $\mathcal{S}$. As in aNET, we iterate the solution of Eq. (7.5)-Eq. (7.10), until convergence (average distance between two consecutive steps of Eq. (7.5) to be $< 10^{-5}$).

7.3.3.3 iNRR vs aNET accuracy

Our model inherits IRM assumption of aNET, hence it also inherits possible inaccuracy due to IRM violation. As aNET vs. iNRR model different ICN architecture, namely on-path vs. off-path caching, their result cannot be directly compared. Thus we evaluate their accuracy against simulation of ⟨SPR,LCE,LRU⟩ and ⟨iNRR,LCE,LRU⟩ respectively, for the small-scale scenario.

As for aNET, we know from [8] that the impact of IRM violation grows with the size of the network under study (or, equivalently, decreases with the density of repository in the network). This is because the IRM assumption does not hold especially for long paths, as miss stream prevails over the exogenous arrivals. As for iNRR, we know from results in the previous section that it possibly significantly shortens the average path length to a cached copy: as such, we can expect IRM violation to affect iNRR less than aNET.

We compute accuracy with respect to simulation for (i) each node individually, as well as for (ii) all nodes having the same distance $\{x : D(x, \mathcal{S}) = d\}$ from the repository. More precisely, indicating the average hit probability for node v as $\bar{\pi}_v$, we evaluate accuracy as the ratio $\bar{\pi}_v^{sim}/\bar{\pi}_v^{model}$.

We consider a 10x10 grid, where the iNRR gain over SPR is visible (recall Fig. 7.4). Consequently, we expect aNET to be negatively affected by the large topology size, as the

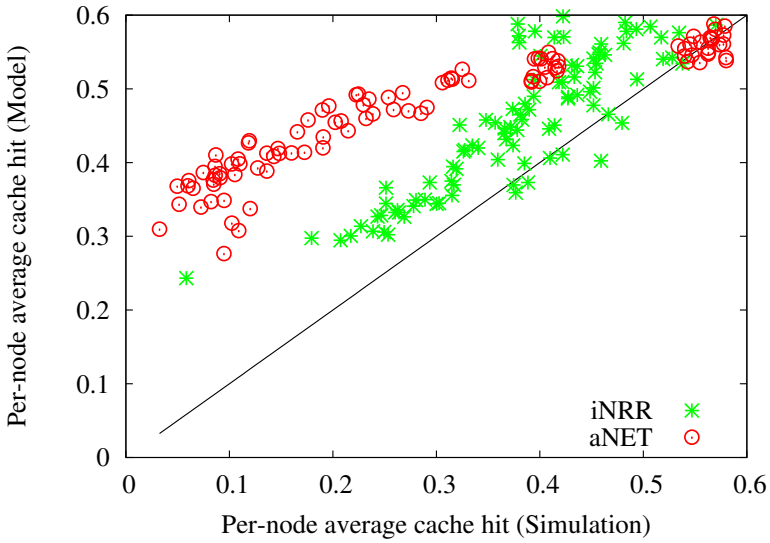


Figure 7.9: Scatter plot of the average cache hit per node $\bar{\pi}_v$ obtained via simulation vs model, for aNET and iNRR, on a 10x10 grid.

SPR distance to $\mathcal{S}$ can grow quite large. Conversely, as iNRR should find closer copies, we expect it to limit the impact of IRM violation (see Sec. III-A of [8]). Secondly, under iNRR nodes split their miss stream across each neighbor: as this mixes independent miss streams flows, it results in a more IRM-like miss flows with respect to SPR routing (similarly to what happens by increasing the k-arity of the SPR tree in Sec. III-A of [8]). Hence, we point out comparison on the same scenario to be unfair, as aNET and iNRR are neither operating on the same distance, nor on the same neighbor fanout. To partly compensate for this bias, we attach clients to each grid node, i.e., $\lambda_{i,v} > 0, \forall v$, so to reinforce the IRM component of the request arrival.

For the sake of readability, in top plot of Fig. 7.8 nodes are ranked for increasing $\bar{\pi}_v^{sim}/\bar{\pi}_v^{model}$ ratios. In the bottom plot of Fig. 7.8, we complement the average ratio with standard deviation bars. First, results confirm that iNRR error is significantly lower than aNET. We can further observe that the iNRR error is less affected by the topological position (essentially, SPR distance) from the repository with respect to aNET[7]. We further show a scatter plot of the average cache hit per node $\bar{\pi}_v$ obtained via simulation vs model in Fig. 7.9, showing that under iNRR model overestimation reduces especially for nodes with low cache hit.

[7]In case of aNET, ratio becomes closer to 1 as the distance from the repository increases: notice the large plateau of about 20 nodes (corresponding to leaves of the SPR distribution tree rooted at $\mathcal{S}$) having unity ratio in top of Fig. 7.8, that are aggregated at $d = 18$ in bottom of Fig. 7.8.

We additionally compare accuracy results on a 4x4 grid and on a binary tree (not shown). As expected, in the 4x4 grid the accuracy of aNET improves (as the difference in the average distance induced by SPR an iNRR forwarding shrinks). In the tree case, iNRR and aNET performance are very close (as iNRR cannot fully exploit nodes neighborhood due to lack of alternative paths, hence miss streams mix less under iNRR).

7.3.4 Approximate iNRR implementation

It should be clear that iNRR is an ideal forwarding policy, requiring an oracle or, equivalently, the knowledge of the state of all caches to instantaneously propagate of in the whole network. We therefore propose two alternative, and practically viable, implementations of Nearest Neighbor Routing (NRR). We cast these solutions on the ground of the general framework we develop in [10, 86], that we briefly recall here.

We assume ICN nodes to be equipped with a FIB data structure, proactively populated by a SPR routing protocol, containing information that allows to follow the shortest path toward a permanent copy of the repository. Requests forwarded along the FIB have thus the chance to find *on path* cached copies, and in case no cached copy is found, they ultimately access the permanent replica stored at the custodian.

Additionally, we require ICN nodes to be equipped with a Temporary FIB data structure (TFIB), reactively populated by an *off path* exploration of the ICN network, triggered by user demand on a new request. We assume that the *exploration phase* is carried only for the first (or few) chunk(s) of a new content, and is aimed at dynamically constructing a path toward the closest cached replica. The path is then stored in the TFIB. In the subsequent *exploitation phase*, the forwarding process can use the new TFIB entry for the next chunks requests of the same content (overriding thus FIB entries). While it is outside the scope, we point out that TFIB is possibly managed as a LRU cache, so that TFIB entries span over subsequent requests of different users for the same content.

In this section, we focus on the *exploration phase*, of which we provide two alternative implementations based on scoped flooding, namely NRR' and NRR", that respectively require one and two phases. Both NRR' and NRR" flood requests over the network, limiting the flooding scope via a TTL field. In modeling terms, NRR limits the radius ρ of the ball centered around v, i.e., $B_\rho(v) = \{u : D(v,u) \leq \rho\}$.

Differences from NRR' and NRR" arise in the way requests are treated during the exploration phase. NRR' floods *regular request* packets, so that it generates possibly multiple data chunks in return – one per each cached copy found in $B_\rho(v)$. Hence, NRR' possibly generates an overhead in terms of load and cache eviction rate, though the duration of the exploration phase is the minimum possible before the closest copy is hit. Conversely, NRR" floods *meta request* packets, with a flag set to indicate that only a binary reply concerning content availability, but not the whole content data, is requested in return[8]. Replies of this

[8]This technique is already commonly used, e.g., in HTTP GET vs HEAD request methods: in the former case, the HTTP response encapsulates the whole object data, in the latter case, only the headers

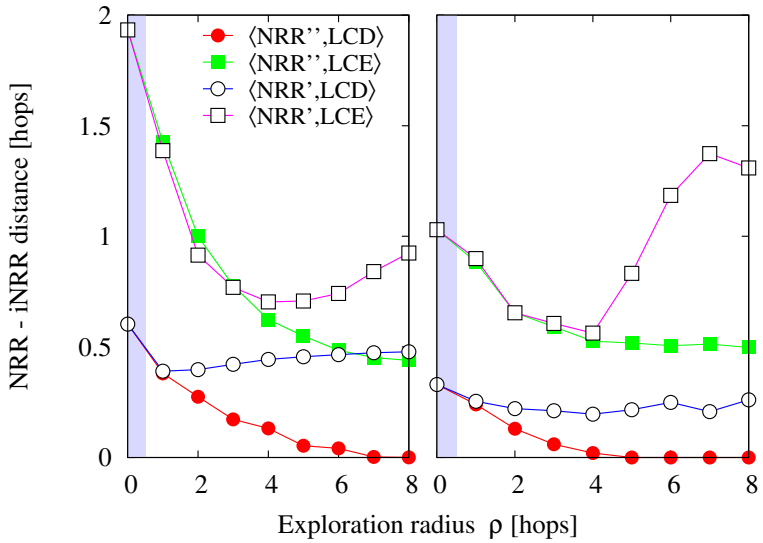


Figure 7.10: Additional distance of NRR implementations with respect to iNRR. NRR' vs NRR" forwarding, for LCE or LCD meta-caching, as a function of the exploration radius ρ. 10x10 grid (left) and 6-level fully redundant binary tree $\mu = 1$ (right).

first phase populate the TFIB with a negligible load and do not force cache eviction (as only meta information about the chunk is sent), but however introduce a delay (as after the first phase the TFIB has been populated, but the content has not been downloaded yet).

Before considering the tradeoff induced by NRR' vs NRR" in terms of load vs delay, let us first analyze their impact on cache eviction. We compare NRR' and NRR" to iNRR by measuring the number of additional hops needed on average to find the content. For completeness, we consider $\mathcal{D} \in$ {LCE,LCD} and $\mathcal{F} \in$ {iNRR,NRR',NRR"} and measure the number of additional hops with respect to the ideal ICN strategy ⟨iNRR,LCD,LRU⟩. Fig. 7.10 depicts the number of additional hops as a function of the radius ρ for the grid (left) and tree (right) topologies. The picture reports all $\langle \mathcal{F}, \mathcal{D} \rangle$ combinations of NRR" (black) vs NRR' (white) and LCD (circle) vs LCE (square) settings. Notice that for $\rho = 0$, NRR degenerates in SPR routing (shadowed region).

Several interesting insights are gathered from Fig. 7.10. First, performance of ⟨iNRR,LCD,LRU⟩ can be approximately arbitrarily close with ⟨NRR",LCD,LRU⟩, as the additional distance goes to zero for $\rho \geq 6$ on trees and grids. Second, cache eviction due to LCE implies an important performance penalty for both NRR' and NRR" (as expected due to results in previous section). Third, cache eviction translates into an important performance penalty

concerning the object.

as well. This is due to the use of regular request packets in NRR', generating data in return that possibly yields multiple cache evictions (even under LCD). Fourth, notice that additional distance decreases for growing ρ only in the case of NRR": this means that NRR" exploration is not only effective but also robust. Conversely, in the NRR' case, whenever ρ increases, eviction increases as well due to both higher chance to find the content on the one hand, and longer paths up to ρ on the other hand. This phenomenon is especially evident for the tree under LCE meta-caching: as soon as ρ becomes comparable with the distance to the repository, this allows a significantly larger portions of the tree to be explored, with consequent massive eviction[9]. We therefore conclude that $\langle$NRR",LCD$\rangle$ with (arbitrarily large) ρ values is able of (arbitrarily close) iNRR approximation.

We now comment on the load and delay induced by NRR' and NRR". As far as load is concerned, NRR" is clearly more lightweight than NRR'. Indeed, while the number of requests sent by NRR' and NRR" is the same, the amount of data chunks sent in return equals either (i) the number of cache hits for NRR', or (ii) the single closest hit for NRR". As chunks travel multiple links, NRR" significantly reduces the load not only because it sends a single chunk (major impact on load), but also because it sends the closest among all cached chunks (second order impact).

As far as delay is concerned, NRR' is possibly faster than NRR" due to the fact that whenever the data is found, it is immediately sent back, whereas NRR" requires an additional phase. While at a first glance it may seem that delay under NRR" would be roughly double with respect to NRR', however this is not the case. Observe first that exploration delay only affects the first chunk, and not subsequent chunks that instead exploit readily available TFIB information. Hence, the delay penalty of the first chunk diminishes weighted over the whole content transmission. Additionally, Fig. 7.10 shows that content is closer in NRR" than in NRR': for instance, in the 10x10 grid, the median number of hops is 2 under NRR" and 3 under NRR'. Denoting with δt the average link delay, the median duration of the two phases in NRR" takes $2(2\delta t)$, while the median duration of the single phase in NRR' is $3\delta t$ (a modest 25% difference, that additionally applies to the first chunk only).

7.4 Conclusions

Through this section, we studied and analyzed forwarding strategies for network caching. Basically we show that: i) exploitation strategies perform bad, as the forwarding is constrained on pre-determined paths; ii) exploration strategies like iNRR, coupled with meta caching produce the best results, with a maximal 60% gain; iii) modeling iNRR lowers the approximation error introduced by neglecting the IRM hypothesis; iv) implementing

[9]Intuitively, under LCE cache pollution extends to the other side of the tree. Under LCD, as popular content is pulled toward the edge of the network, requests do not explore the whole network, successfully limiting cache pollution.

iNRR by the means of meta requests does not introduce caching pollution, and rapidly approaches to the ideal iNRR forwarding.

Under this light, though [2] dismisses ubiquitous caching due to its limited gains, it nevertheless misses the most important piece of the puzzle – namely the meta-caching policy. Our results show indeed that ⟨iNRR,LCD,LRU⟩ can obtain significant gains beyond the ⟨iNRR,LCE,LRU⟩ strategy advertised in [2] as the ICN optimum. Clearly, business considerations will answer whether such gains are economically worth the deployment of ICN – yet, business considerations should be taken on the ground of all relevant technical information.

Chapter 8

ccn$\mathcal{S}$im: an Highly Scalable CCN Simulator

In this conclusive chapter, we describe ccn$\mathcal{S}$im, our tool used throughall the second part of the thesis. We start by doing a description of the available CCN/ICN software and then show the internal of ccn$\mathcal{S}$im, describing its architecture and performance.

8.1 Taxonomy of ICN Software

We report in Tab. 8.1 the list of ICN open source software we consider in this census: to the best of our knowledge, the list includes all publicly available software. We instead purposely exclude scientific publications available in the literature whose evaluation is based on software that is *not* publicly available.

As previously anticipated, about half of the available software pertains to different ICN proposals (top portion of Tab. 8.1), while the other half pertains a specific ICN proposal (namely, CCN [41], bottom of Tab. 8.1), that we treat separately in this section.

For each software, the table reports the latest release and its date, that correlates with the development activity (notice that some software reports no explicit tags for its release date, for which we inspect the file headers). The table then reports the language used by the core library (that correlates with the startup cost) and the additional languages for accessory software (either bindings of the main library API for a number of scripting languages, or languages used to develop the accessory applications). Operating system upon which the software have been tested (or mentioned to run) is reported next, and represent a potential barrier to development. Finally, the type of software and the project under which the software have been released are mentioned. We believe that architectural properties will guide the selection of a specific ICN paradigm. Yet, we argue that, after a specific ICN paradigm has been selected, the startup cost will be an important factor in letting new user select between potential software alternatives – to which we thus pay

Table 8.1: Taxonomy of ICN and CCN open source software

Scope	Software (and URL)	Latest Release	Release date	Language Core	Language Extra†	Operating System	Sofware type	Project
ICN	MobilityFirst [108]	*none*	Exp. 08/2012	C++	-	Linux	Click prototype	[109]
	Blackhawk [110]	0.3	06/2010	C	P,R	FreeBSD	Prototype	[111, 112]
	Blackadder [113]	0.3.1	09/2012	C++	P,R,J	Linux	Click prototype	[112]
	ICNsim [114]	n.a.	06/2012	C++		Linux	Omnet++ simulator	[115]
	NetInf (nilib) [116]	0.2	10/2012	C	P,J,R,PH,cl	*NIX	Prototype	[117]
	openNetInf [118]	1.0	10/2011	Java		Portable	Prototype	[117, 119]
	PeerKit [120]	-	12/2011	Java		Portable	Prototype	[121]
	Conet [122]		12/2012	C		Linux	Prototype	[121]
CCN	CCNx [123]	0.7	12/2012	C	J	Linux, Android	Prototype	[124]
	ns3 CCNx [125]			C	P	Linux	ns3 CCNx simulator	[126]
	NDNsim [127]	n.a.	*05/2012*	C++	P	Linux, MacOS	ns3 Simulator	[124]
	ccnpl-sim [128]	0.1	11/2012	C++		Linux	Custom simulator	[126]
	ccnSim [101]	0.1	03/2012	C++		Linux	Omnet++ Simulator	[126]

†Extra language legend: P=Python, J=Java, PH=PHP, R=Ruby, cl=clojure

close attention in the following.

As a final remark, we voluntarily kept the table simple to allow to grasp the ICN software census at a glance. As a result, the table is missing some relevant information (such as the software license), as well as other information that though relevant (such as the amount of lines of code, the existence of proper documentation, etc.) could quickly become outdated.

8.1.1 ICN software

The scope of the ICN software can be further divided into classes according to the project they pertain to: namely, tools can be reconducted to either the MobilityFirst [109] project[1], or the PSIRP [111] and PURSUIT [112] project suite, or the 4WARD [119] and SAIL [117] suite, or the CONVERGENCE [121] project[2].

We instead omit from the list projects such as COMET [129] or COAST [130] that, although closely related in scope, have no explicit commitment in releasing their software tools as open source to the scientific community. As previously outlined, most of the software released in the ICN scope represent prototype implementation of a reference ICN architecture. Notice that it is possible that multiple software implementations of the same architecture is released by the same (or followup) projects as for the Blackhawk/Blackadder implementations of the PSIRP/PURSUIT project suite, or openNetInf/nilib implementations of the 4WARD/SAIL NetInf suite. We now briefly overview each software and

[1]Remark: although the MobilityFirts project has not released any software to date, we include it in the list since it has already been demonstrated at multiple venues [109] and the software release was expected during August 2012 and is thus likely only delayed.

[2]The knowleadgeable reader will notice that though CONVERGENCE builds over CCN, it also modifies some aspects with respect to its original proposal [41], which is why we prefer to include it under a larger ICN umbrella.

compare alternative software in the same class.

PSIRP/PURSUIT Blackhawk is a prototype implemented in C that integrates the publish/subscribe system directly in the FreeBSD kernel. To reduce the duration of the initial learning phase, wrapper for the low level library are provided in several high-level scripting languages, such as Python and Ruby, that are obtained through SWIG [131]. Similarly, to lower the initial adoption barrier, ready to use FreeBSD virtual images running Blackhawk are provided for KVM, VMware or VirtualBox. However, the most recent Blackhawk software was relased on June 2010, and the PSIRP/PURSUIT development was continued in Blackadder. Most notable changes are the implementation as a Click [132] modular router component (supporting both kernel and user mode), using C++ on Linux (instead of C and FreeBSD), the addition of some wrappers (e.g., Java and C), and applications (a pub/sub video application based on VLC).

An important side-effect of Blackadder implementation as a Click modular router component [132] is worth highlighting: indeed, though the software is primarily ment to be a full-blown prototype, it follows that is possible to run simulations through the Click vs ns2/ns3 integration, that are known as *nsclick* and *ns-3-click* extensions [133]. However, further effort [134] in the PSIRP area seems to suggest that this approach lacks of the necessary flexibility and involves a non negligible overhead – since, by hacking low level calls of the OS networking stack the actual prototype code is run in a simulated environment, development is significantly more complex with respect to simpler abstractions (and dirty hacks) commonly used in simulation. This motivated the development of an Omnet++ based simulator ICNsim [114], that follows the publish-subscribe approach introduced by PSIRP. ICNsim is developed in the context of the PAL [115] project, that is aimed at offering personal health-care services. ICNsim does not aim at faithfully representing the whole PSIRP architecture and instead mainly focuses on the topology manager design, to assist the design of network path computation algorithms. As for the scalability properties of ICNsim, they remain unclear in that [134] only consider a toy-case topology consisting of 12 routers, 4 publishers and 4 subscribers for the sake of the example. Additionally, the use of the INET/MANET frameworks suggests the simulator to perform operations at packet level, entailing that simulation are performed at the finest possible grain.

4WARD/SAIL The 4WARD [119] and SAIL [117] projects have released two open source implementation of their Network of Information (NetInf) architecture. openNetInf [118] is the oldest between the two, is implemented in Java and therefore portable over a number of platform (including Microsoft Windows), was released during [119] and last updated on October 2011. The core library has a significant amount of documentation and is complemented by plugins for popular applications, such as Firefox and Thunderbird, that let browse NetInf enabled servers and send

emails to Information Object (IO) identifiers rather than email addresses respectively. As for Blackhawk, openNetInf is also available as an Ubuntu-based Linux virtual image for VirtualBox. Interestingly, open testbed facilities are available, with 3 nodes (`nn1.testbed.netinf.org` ...`nn3.testbed.netinf.org`) running the openNetInf software that can be publicly accessed instead of setting up a local NetInf node for testing purposes – which can lower startup cost significantly.

More recently, nilib [116], a C implementation of NetInf with wrappers in Python, Java, Ruby and clojure was released in October 2012. Notable difference with respect to openNetInf is the implementation of the Named Information (NI) URI schemes including truncated hash suites, binary vs human-readable name format under discussion at the IETF [135, 136]. As the authors explicitly state, the library has been used in interoperability tests, though the code is work in progress and the ecosystem surrounding it is at time of writing, less mature with respect to the openNetInf one (e.g., absence of virtual machines or testbed). At the same time, with openNetInf no longer active developed[3] and with nilib software releases about every quarter, it is likely that the latter ecosystem will soon reach the same level of the former.

CONVERGENCE Finally, the CONVERGENCE [121] project has released prototype implementations of its architecture, available in two software packages, namely PEERKIT[4] [120] and COMET [122] (the latter partly in cooperation with the OFELIA [137] project).

In more details, PEERKIT [120] is a (Java) middleware extension of the new MPEG-M standard to support distributed applications that create, trade and consume digital objects. Network level functionalities are provided by the CONVERGENCE Network (CoNet), that embraces a publish/subscribe paradigm inspired on CCN, but that also remains backward compatible (e.g., supporting "un-named-data"), takes an evolutionary approach (i.e., it is integrated in existing IP networks by using a new header option [138]) and uses alternative lookup methods based on a name-system (with respect to a full dissemination of names via a routing protocol as in CCN). The released software includes a simple sample application that allows for a simple photo/document sharing.

Finally, in cooperation with the OFELIA [137] project that focuses on Software Defined Network (SDN), and specifically aims at providing OpenFlow-based experimental facilities publicly and free-of-charge, a lower-level implementation of the CoNet architecture has also been released. Namely, CONET [122] is implemented in the

[3]The mercurial branch has been tagged as v1.0_prePg3 and v2.0_pg3_final on Nov. 4th, 2011.

[4]Remark: to download the middleware, beware that the project download page points to a non-existing URL `http://147.102.9.7/convergence_files/cpk/CPK.zip`. With simple trial and error we have determined the correct link to be [120].

OpenFlow 1.0 framework [139], and is heavily based on the CCNx 0.6.2 prototype[5] that we cover in the next section. Notice that the implementation is not straigthforward, since while CONET provides content-name as IP option in the IPv4 header, however the version 1.0 of the OpenFlow implementation cannot parse IP options, so that workaround are necessary as described in [140]. The software has been first released on December 2012, with (thin) instructions on how to setup a CONET demonstration, but the codebase will likely mature as projects enter their final year.

8.1.2 CCN software

Among the many ICN architectures sprouted in the last decade, the one which has received, by far, the largest attention is the Content Centric Networking (CCN) paradigm [41]. This is also testified by the greater diversity of tools available in the CCN context, that explore different operational points in simplicity vs realism design space (or, equivalently in the scalability versus complexity cost space), as schematized in Fig. 8.1.

Shortly, if we were about to describe each CCN-related software[6] with a single keyword, we might say that CCNx [123] aims at *deployment* and experiments, ns3 CCNx-DCE [125] at *faithful* simulation, ndnSIM [127] at *completeness*, CCPN-Sim [128] at *fine-grained* packet-level simulation and ccn$\mathcal{S}$im [101] at *scalable* chunk-level simulation. Notice that, although we have developed one of such tools, our view of the usefulness of each tool is not biased. Rather, we argue that each of these tools, that we briefly describe, fills a necessary gap in the spectrum of Fig. 8.1.

CCNx In more detail, CCNx [123] is a fully operational prototype of CCN. Low level portion of the prototype are written in C, with ongoing effort to port part of them (e.g., forwarding) in kernel-space. New applications that run natively over CCN being continuously developed in Java and C. Example applications include a simple text chat tool (that allows users to communicate and is often used for testing connections and managing testbed operations) and an audio conferencing tool. An Internet-wide testbed [142] building over CCNx (sending CCNx chunks over UDP tunnels) is available, that is planned to be open on the long term but is now available for NDN partners only the time being (due to substantial on-going work on routing, forwarding, and network management that currently requires more control over the topology and users).

However, CCNx does not provide per se any testing, validation or experimentation capabilities. As such, users have to instrument ad hoc local testbeds or use dedicated

[5]Remark: it is unclear to what extent CONET should be considered as a fork from the CCNx prototype, and to what extent (or at which complexity) it could benefit from later updated of the CCNx codebase.

[6]Though another Omnet++ simulator of CCN was presented at CCNxCon'12, neither the code is however not available, at time of writing, nor an URL is available in the presentation [141], so we do not consider in this chapter.

infrastructures such as Grid5000 [143] or PlanetLab [144], which is a rather complex task. As such, a preliminary simulation step is required in order to understand system dynamics and define and select the best performing algorithms to implement in the prototype.

ns3 CCNx-DCE An intermediate step toward the above solution is represented by ns3 simulation with Direct Code Execution (DCE) [125] of the (opportunely recompiled) CCNx prototype. Shortly, the DCE ns3 module provides facilities to execute within ns3 existing implementations of user-space and kernel-space network protocols. For instance, the Quagga routing protocol and recent versions of the Linux network stack run under DCE, so that faithfulness to real-world implementation is not only guaranteed for the CCNx prototype but for lower layers of the networking stack as well.

This approach has the advantage of using the same codebase of CCNx and additionally simplifies the experimentation. Hence, ns3 CCNx-DCE can be seen as a mandatory step for anybody having a strong commitment to do CCNx prototyping, greatly simplifying the development phase and assisting debugging, troubleshooting and (small-scale) performance evaluation of the CCNx prototype.

At the same time, maintaining a single codebase is however not without cost, especially in terms of a greater development effort – as it is rather complex to modify the prototype to explore different design approaches. Furthermore, as the *whole* CCNx stack is run (e.g., including crypto due to security), this hinders scalability of the simulation. Alternatively, to avoid incurring in this performance penalty, the CCNx prototype might be opportunely tweaked so to turn on/off some features that are supposedly irrelevant for the aspect under study[7]. For these reasons, simpler simulator environments –such as those introduced next– may be preferable for anyone that does not necessarily plan to invest a possibly significantly amount of manpower in CCNx prototyping.

ndnSIM Another option is to perform ns3 simulation with the ndnSIM [127] simulator – notice that patches to ns3 are required to run ndnSIM. The primary goal of ndnSIM is completeness, and the design of ndnSIM follows the philosophy of network simulations in ns3, which devises maximum abstraction for all modeled components. A thorough description of ndnSIM components is available in [145]. Basically, ndnSIM faithfully ports CCNx implementation in a modular ns3 simulator, which also means that the number of currently implemented policies is small (e.g., ndnSIM only provides LRU, FIFO and random cache replacement policies [145]).

At the same time the design makes easy to intercept and modify the default behavior – as every step of an Interest and Data packet handling, including Content Store,

[7]Though regression tests are needed in order to factor out possible unexpected protocol behavior due to some features being turned off, which may not payoff the cost.

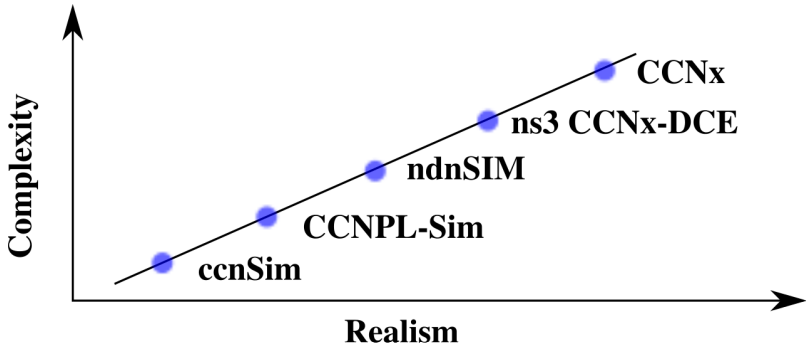


Figure 8.1: Simplicity vs realism in CCN-related software

PIT, FIB lookups, is represented as a virtual function, they can be overridden by user-defined classes. Yet, completeness does not comes without cost – as e.g., storing metadata associated to chunks requires about 1KB of RAM memory [146], which may still pose scalability problems. Hence, ndnSIM represents a fairly good compromise between completeness (and it thus may be a good fit for many CCN aspects including security) and scalability (though slightly more inclined to the former).

CCNPL-Sim CCN Packet Level Simulator (CCNPL-Sim) is a packet level CCN simulator. It is based in part on CBCBsim [147] from which it imports part of the forwarding layer and the name-based routing protocol, namely the Combined Broadcast and Content-Based (CBCB) routing scheme [148], while the CCN protocol features have been developed from scratch. The simulator has been used to model the per-hop forwarding behavior of CCN nodes [149] and the receiver based congestion control protocol tuning interest ingestion [150], where a fine-grained control over individual packets is imperative to get accurate performance results.

Differently from CCNx-DCE and ndnSIM (both based on ns3) and from ccn$\mathcal{S}$im (based on Omnet++), CCNPL-Sim has the drawback of using a custom discrete-event simulator, that is unfamiliar to most of the researchers and thus could require an additional learning cost. Additionally the mandatory usage of CBCB narrows the possible experimentation area, making it CCNPL-Sim unfit for, e.g., routing studies (for which ndnSIM is a better fit) or large scale studies (for which ccn$\mathcal{S}$im is a better fit).

ccn$\mathcal{S}$im ccn$\mathcal{S}$im is a scalable chunk-level CCN simulator, written in C++ under the Omnet++ framework, and allows to assess CCN performance in scenarios with large orders of magnitude for CCN content stores (up to 10^6 chunks) and Internet catalog sizes (up to 10^8 files) on off-the-shelf hardware (i.e, a PC with a fair amount of RAM). ccn$\mathcal{S}$im focuses on the performance of caching replacement, decision and forwarding

policies, and neglect routing and security aspects (for which ndnSIM is a better fit) or congestion control policies (for which CCNPL-Sim is a better fit).

Hence, a large number of policies is implemented with respect to e.g., ndnSIM (that only provides LRU, FIFO and random replacement policies and shortest path routing [145]), that allow to explore wider boundaries of the design space. Moreover the naming scheme is not faithfully represented, which on the one hand limits the usefulness of ccn$\mathcal{S}$im for, e.g., routing studies, but on the other hand greatly enhance the scalability – as the memory footprint of individual PIT and CS entries is about 64bits, which is about 2 orders of magnitude smaller with respect to ndnSIM. We describe ccn$\mathcal{S}$im in details in Sec. 8.2 and offer a thorough analysis of its performance in Sec. 8.3.

8.2 Description of ccn$\mathcal{S}$im

ccn$\mathcal{S}$im was designed aiming at scalability, for which we traded off realism in favor of simplicity. Our work was motivated by the fact that, while ICN and CCN is expected to serve Internet wide content, the majority of performance evaluation studies were only able to deal with rather simple and small-scale scenarios, especially in terms of dominant factors like catalog, network topologies, and cache size (see Sec. 6.1. For instance, the YouTube catalog is estimated to be on the order of 1 PB i.e., 10^{15} Bytes (details in Sec. 8.3.1): yet, largest catalogs considered in the literature are off by some orders of magnitude (i.e., 20K object or 138GB [4]). Similarly, while [44] sizes to about 10 GB the amount of memory that can be addressed at line speed in ICN architectures, current simulations operate with much smaller caches (e.g., 6.4MB [92]-50MB [4]).

Over the last years, we have developed and optimized ccn$\mathcal{S}$im, an highly scalable chunk-level simulator especially suitable for the analysis of CCN caching performance, that we have released to the scientific community as open source software [101]. As we will briefly overview in the following, ccn$\mathcal{S}$im is a modular, flexible, and fast CCN simulator, capable to deal with different caching algorithms/policies, forwarding strategies, topologies, and popularity laws. Specifically, two crucial aspects have been taken into account in the design of ccn$\mathcal{S}$im: namely, (i) memory occupancy, in reason of the size of caches and catalog and (ii) CPU time, in reason of per-chunk operations.

While in previous chapters we focused on the performance of CCN gathered through ccn$\mathcal{S}$im, in the context of this chapter we rather focus on the implementation (this section) and performance (Sec. 8.3) of the ccn$\mathcal{S}$im tool itself.

8.2.1 Simulator architecture

Basically, ccn$\mathcal{S}$im is a C++ package built on the top of the Omnet++ framework [151]. In the following, we give an overview of the set of ccn$\mathcal{S}$im classes, that are illustrated in

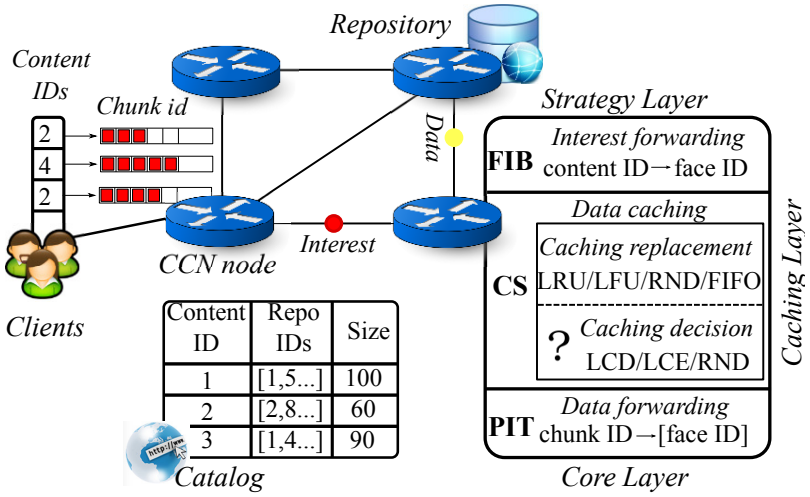


Figure 8.2: ccnSim main components at a glance.

Fig. 8.2, highlighting the key design choices.

8.2.1.1 Catalog and popularity model

Clients represent an aggregate of users who request contents with a given popularity distribution. Requests for new content follow a Poisson process, with a customizable rate. In ccnSim, users are not CCN nodes, thus they do not implement the whole CCN stack. Each aggregate is implemented as an STL multi-associative map[8] that keeps track of different contents downloaded by individual users.

Popularity model, and hence catalog and content size, represents a crucial aspect of every ICN architecture. We have designed and accurately tuned the catalog and popularity model according to recent literature in Internet metrology (details in [9]). Size of the files in the catalog is a configurable parameter, and follows a geometric law by default [46]. For matters of efficiency, the size of each file is computed during the simulation bootstrap, and is stored within a large static array (named *catalog* in Fig. 8.2).

We model popularity distribution with a Mandelbrot-Zipf (M-Zipf), shaped by parameters (q, α), that we statically initialized during the startup of the simulator, and we employ a $\log(N)$ binary search for each M-Zipf random number (with N size of the catalog). Notice that while this may seem a minor implementation detail, a binary search rather becomes a requirement when the catalog size reaches $N = 10^8$ objects as for YouTube [48]. The popularity model possibly includes a spatial heterogeneity to account for skew in the popularity

[8] A multimap is like a simple map, but it considers the possibility of having more entries with the same key.

law due to geographical/cultural barriers [106].

8.2.1.2 Messages and chunks

At low level, interests and data messages carry a 64-bit unsigned integer, namely the *chunk identifier*. This is a simplification with respect to the original naming structure of CCN, that however gives ccn$\mathcal{S}$im an advantage of several orders of magnitude in terms of memory scalability (e.g., PIT and CS entries consume about 64bits in ccn$\mathcal{S}$im with respect to about 1KB in ndnSIM [146]).

The chunk ID stores information about the content name (most significant 32 bits) and the chunk sequence number (least significant 32 bits). This design represents a tradeoff between space and flexibility. On the one hand, we minimize the space needed for moving content all over the network. On the other hand, we have 32-bit content identifiers (up to 4 billions of individual contents).

8.2.1.3 Node architecture

This simulator is not targeted for a particular topology, and the user can freely arrange the nodes following her needs. However, we provide 8 built-in topologies: five of them are realistic ISP networks (Geant, Abilene, Level3, Qwest, Sprint), and three are synthetic (random, torus and tree). For each network, ccn$\mathcal{S}$im is able to build different routing FIBs (either precomputing the shortest-path and multiple-path based on standard graph algorithms [9] or possibly with a dynamic flooding-based exploration(see Sec. 7.2). As we can observe from Fig. 8.2, a CCN node comprises three different submodules: core, cache, and strategy layer.

Core layer. Is the responsible for PIT management, and communicates with caching and forwarding layers. Any new interest raises a CS lookup. If the data is not in the CS, but a repository for the given content is attached to the node, the CCN node returns the data packet back. Otherwise, it deflects the interest to the forwarding layer. Any data chunk raises a PIT lookup, that eventually sends the content back on all the PIT interfaces for that chunk. The PIT is implemented as an associative map of arrays, indexed by the 64bit chunk identifier.

Caching layer. Caching is one of the crucial aspects of any ICN architecture. CS acts according to a caching *decision policy* (i.e., whether to store the data in CS or not) and a *replacement policy* (i.e., what to drop from CS in case it is full). While we already provide a fairly large number of decision (LCD [16], Random [19], LCE) and replacement policies (LRU, Random [45], FIFO) we have designed the CS in a modular fashion. Specifically, new replacement algorithms are implemented as modules overwriting the caching polymorphic methods `store()` and `lookup()`; a similar trick is done for new decisions policies with the

polymorphic method `isToCache()`.

At low level, CS is an associative map. Since CS lookups are very frequent operations in CCN, we optimized their implementation. In fact, while naive LRU implementations can represent a drastic bottleneck in large-scale simulation, we resort to an efficient LRU implementation through a map of pointers [44] (the map speeds-up the access to the elements in cache, while pointers are used to take the elements order).

Strategy layer. The strategy layer basically takes decisions about interest forwarding, through the `getDecision()` method. This polymorphic function returns a bit mask with the same cardinality of the output interfaces set. A 1 in the i-th position of the mask will forward an interest toward the i-th interface.

The default strategy layer in ccn$\mathcal{S}$im sends messages toward the nearest repository over the shortest path (to speed-up simulation, FIB of each node are pre-filled at the simulation startup and `getDecision()` spoofs from the catalog the repositories who store permanent copies). Other multi-path strategies, both static (see Sec. 7.2) and dynamic (see Sec. 7.3) are already available, while further strategies can be implemented by overwriting the `getDecision()` method.

8.2.1.4 Simulation statistics

Statistic collection starts only when the cache hit metric has reached a stationary state. In more details, we start with empty caches and, as soon as caches fill up, every node samples its hit rate every t_s simulated time (usually t_s is about hundreds of milliseconds), and the variance of the collected samples is computed every t_w (usually t_w is about tens of seconds).

Only when the cache hit variance falls under a given threshold, the node declares itself stabilized. Statistic collection starts only after *all nodes are stable*. The stationary state is then simulated for a customizable duration (typically one or two hours of simulation time) after which statistics of interest are collected (e.g., hit rate, distance, cache diversity, download time, and so forth).

8.3 Benchmarking of ccn$\mathcal{S}$im

We now extensively benchmark ccn$\mathcal{S}$im, considering a challenging YouTube-like scenario, that we describe in Sec. 8.3.1. Our benchmark focuses on two main aspects, that possibly become a bottleneck in terms of simulation feasibility and duration:

- *Memory occupancy*: the simulation of very large catalogs and very large cache size is essential to gather CCN performance under realistic settings. Yet, the CDF of commonly used (i.e., Zipf-based) popularity distributions cannot be expressed in a closed form: hence, huge catalogs swell up the memory demand of the simulator.

- *CPU time*: system dynamics have to be represented at *chunk level*, i.e., a minimal data unit, which is typically packet-size or slightly larger (e.g., 10KB). Clearly, efficient operation require a careful engineering and optimization of the most computationally intensive tasks CCN has to deal with.

Our evaluation is structured along two main axis. The first axis goes along a *profiling of the simulator*, in order to pinpoint the function call representing the major CPU bottleneck. Based on the profiling results, we refactory part of the code to reduce the execution time as much as possible (Sec. 8.3.2).

The second axis investigates *parallel execution* using Message Passing Interface (MPI) capabilities. The main driver here is the fact that, even though the simulator has been designed to have a small RAM memory footprint, the execution of large-scale simulation (in terms of the catalog, network and cache size) will nevertheless sooner or later pose a RAM bottleneck. Since the typically available off-the-shelf servers have a large number of cores (typically 4-8, if not more), it is worth investigating whether the simulation run can be speed-up by parallelizing the execution of multiple cores (Sec. 8.3.3).

Finally, we investigate the scalability of the simulation in terms of the network size, always under the challenging YouTube scenario, developing a simple model of the requirements in terms of memory and execution time that ccn$\mathcal{S}$im user can expect (Sec. 8.3.4). Overall, our careful engineering of ccn$\mathcal{S}$im allows to simulate very large scale scenarios in a reasonable time: to give an idea of ccn$\mathcal{S}$im performance, a common off-the-shelf PC equipped with 8GB of RAM memory is able to simulate 2-hours of a 50-nodes CCN network, where each node is equipped with 10 GB caches, serving an Internet-like 1 PB catalog in about 20 min CPU time.

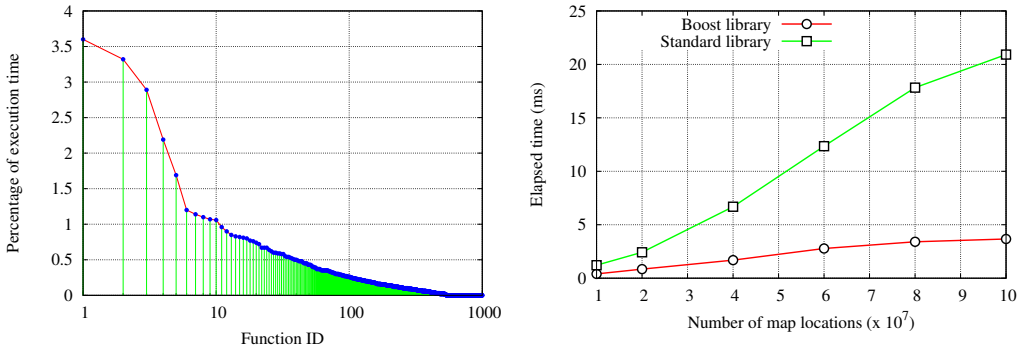
8.3.1 Benchmark scenario

As target application for our benchmark, and in reason of the growing importance of video application in the future Internet, we consider one of the most popular VoD application nowadays, namely YouTube. Further details and motivations with respect to the choice of scenario parameters, that we briefly recap below, is given in Ch. 6.

The YouTube catalog is sized at about 1 PB, as it consists of about 10^8 files [48] having geometrically distributed size with average 10MB [46]. Contents are partitioned in 10KB chunks, thus the average file size in chunks turns to be D=1000 chunks.

As we are not interested on the CCN performance, we select a synthetic torus topology, and let the network size grow from 10-50 nodes (specifically, from 3×3 to 7×7 torus). Nodes are equipped with 10GB [44] (or 10^6 chunks). The decision/replacement pair considered here, is LCE/LRU (the most used within the ICN proposals). As for the strategy layer, interest packets are forwarded toward the nearest repository along the shortest path.

Clients aggregate are attached to each node, and users request have average arrival rate of $\lambda = 20$ Hz. We instrument ccn$\mathcal{S}$im to simulate $T = 2$ hours of simulated time after the cache hit rate stabilizes. Overall, the number of per-chunk operation in steady state can



(a) (b)

Figure 8.3: ccn$\mathcal{S}$im profiling. Functions ordered by decreasing execution time(a). Access time comparison of STL and Boost libraries(b).

be evaluated as ρDTd, with d average path length, and is thus on the order of $[10^8, 10^9]$ depending on the specific scenario.

Results reported in this chapter are gathered on a off-the-shelf server equipped with Intel Xeon E5620 8-cores CPU (running at 2.4GHz), and with 12 MB L3 cache and 24 GB RAM memory.

8.3.2 Simulator profiling

For profiling, we used the standard GNU profiler *gprof*, whose results are shown in Fig. 8.3 – where for the sake of the illustration we show only the first thousand functions. In the picture, functions are ranked by decreasing execution time percentage. For instance, from Fig. 8.3 is easy to see that if the total execution time is 100 seconds, the CPU has been busy with function having rank 1 for about 3.5 seconds. We see that the curve is slowly decreasing for most of the ccn$\mathcal{S}$im functions: this means that each function is individually accounting for only a small fraction of the total execution time. This is a bad scenario for optimization, as after profiling, one would reimplement only the few functions that are representative of the bulk of the CPU time. However, Fig. 8.3 shows that such "quick win" approach does not apply to ccn$\mathcal{S}$im.

Still, the very first handful of functions are responsible for about 15% of the CPU time. A more in-depth inspection, reveals these functions to be responsible for providing access to CCN data structures (CS, PIT, FIB, catalog, etc.) that are implemented as associative maps, array, and so forth in C++. As these structures are accessed very often in CCN (basically, most chunk-based operations will require multiple accesses into the structures, on multiple nodes), we can reduce execution time by adopting their most efficient C++ implementation.

We thus consider the two most common libraries to implement such structures, namely the Standard Template Library (STL) [152] and the Boost library [153]. The comparison is shown in Fig. 8.3(b), that plots the elapsed time of a stress-test program filling the associative map with different integer sequences. Even for very simple operations, the Boost library outperforms the STL of a factor of 4-5. This performance gap is due to the fact that while STL maps are ordered and implemented through a red-black tree, Boost maps are fully unordered and implemented through more efficient hash functions. Apparently, the Red-Black tree maintenance introduces a significant overhead especially for large structures.

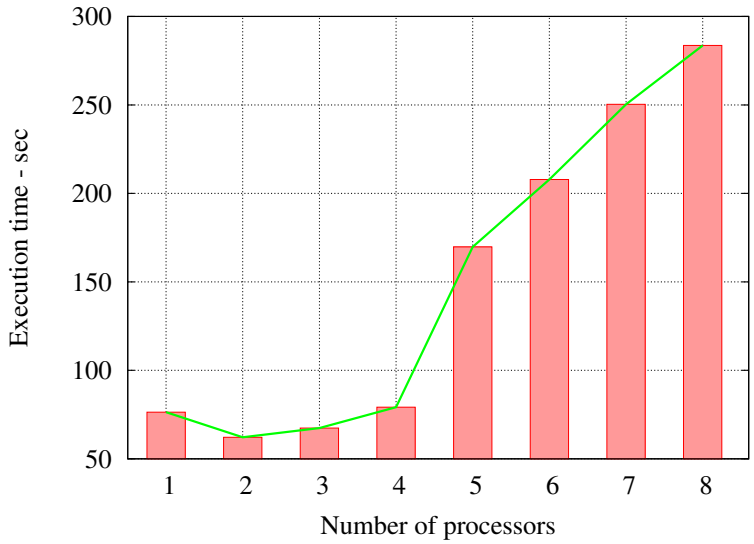


Figure 8.4: Simulation duration vs parallelism degree.

The comparative experiments let us conclude that Boost is more efficient with respect to the STL implementation we were using so far. We therefore refactor ccn$\mathcal{S}$im code using unordered Boost hash-map for any associative map. Notice that while there is a factor of 4-5 speedup in using Boost, this will affect only the first handful of functions related to data structure access. Since these were accounting for about the 15% of the CPU time, we can expect the overall CPU time after refactoring with Boost to be about 85% of the previous execution time under STL.

8.3.3 Simulator parallelization

A Parallel Discrete Event Simulation (PDES) has basically two meanings: distributing the model over *different computers* as a meaning to reduce the memory occupancy of the

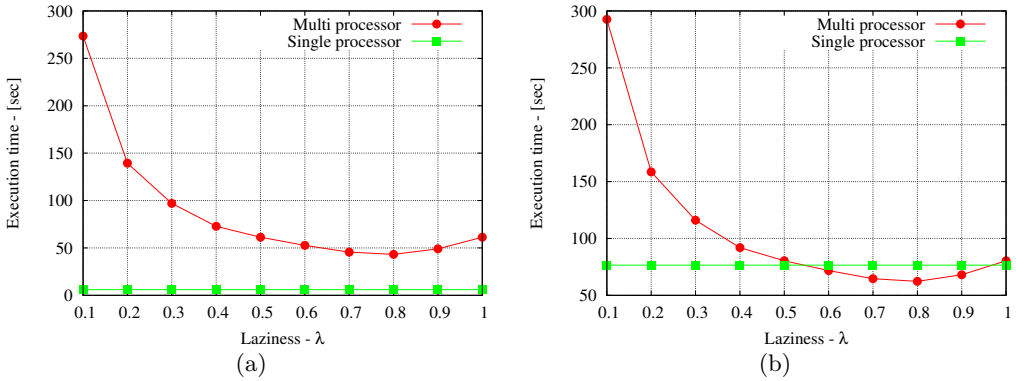


Figure 8.5: Simulation duration vs vs laziness in case of cold (a) and hot (b) startup phase.

simulator; or distributing the model over *multiple processors* for optimizing its execution time. In our case, we're interested in pursuing the second goal only, since otherwise RAM may become a bottleneck, leaving many core possibly unused.

Omnet++ has native support of PDES through Message Passing Interface (MPI). As MPI is a built-in feature, changes in the underlying ccn$\mathcal{S}$im code are minimal. Among different parallel algorithms supported, especially worth of interest is the Ideal Simulation Protocol (ISP) introduced by [154], as it helps to determine the maximum speed-up achievable by any PDES algorithm for a particular model and simulation environment.

We start our experiment plotting the simulation duration as function of the number of parallel processors over which the simulations are split up. We set ccn$\mathcal{S}$im for simulating half an hour of a simpler CCN network (w.r.t. the one described in Sec. 8.3.1) after the transient period ended. Results are shown in Fig. 8.4, and are rather counterintuitive. In fact, performance *significantly worsens* when we increase the number of parallel processors.

To explain these results, we have to briefly introduce PDES concepts of *lookahead* and *laziness*. The *lookahead* is associated with the ability of a logical process (LP) to predict its future behavior: at any simulation time t, if an LP can predict that the earliest event it will cause to occur in another LP is no sooner than $t + \ell$, its lookahead turns to be ℓ. Intuitively, we can say that a small lookahead value badly affects the performance of the overall PDES system, as LPs have to very often synchronize among themselves. The system *laziness* is then correlated with the frequency at which synchronization messages are exchanged between different LPs. In more detail, the laziness is indicated with $\lambda \in [0, 1]$ and represents the synchronization rate, with maximum (minimum) synchronization rate is achieved for λ equal to 1 (0). Generally, a rule of thumb in PDES is to roughly approximate the synchronization period with the system lookahead.

In Fig. 8.5(a,b) we plot the simulation duration, with (b) and without (a) a cache warm-

up phase, as a function of the laziness λ. The plot shows that increasing λ (i.e., increasing the synchronization rate) ccn$\mathcal{S}$im performance tends to ameliorate. At the same time, the number of synchronization messages grow with λ: therefore, increasing λ increases the percentage of time that each individual CPU devotes to the synchronization task, which can also turn into an excessive overhead for large λ (see that execution time increases for $\lambda > 0.8$).

The most important takeaway from Fig. 8.5 is however that in most case, single-processor execution of ccn$\mathcal{S}$im is more efficient than its multi-processor counterpart. Specifically, only minimal PDES gain can be observed when $\lambda \in [0.6 - 0.9]$ and only the hot-startup case of Fig. 8.5(c). Hot-startup means that CCN caches are pre-fill at time $t = 0$ with random content (proportionally to the catalog popularity), while in the cold-startup case CCN caches are empty at $t = 0$. Since cache warm-up does not need synchronization (as each cache is independent), the warm-up phase may result in shorter execution time in PDES (at least for some laziness values). However, gains are ephemeral since cache warm-up can possibly induce longer convergence time of the cache hit metric in case of a mismatch between the initial cache content and the replacement policy[9] Conversely, with cold-startup of Fig. 8.5(b) the single processor is always better than PDES. We conclude that the bulk of the CCN operation requires high synchronization frequencies, which entails that CCN simulation has an inherently small lookahead ℓ and is thus inherently non parallelizeable.

8.3.4 Overall performance

Finally, we report the expected simulation time and memory occupancy for large-scale CCN simulation under the scenario described in Sec. 8.3.1. Fig. 8.6(a) plots the running time of a simulation, as function of the network size. We can breakdown the total simulation duration t_{tot} (i.e., the CPU time) as the sum of the *bootstrap* time t_B (e.g., fill the catalog, build the network, allocate data structures, build reverse index to speed-up lookup, etc.), plus the cache *fill* time t_F (especially important in case of cold startup with empty caches), plus a *transient* period until the cache hit rate reaches a steady state t_T, plus the CPU time t_S needed to run 1 hour of simulated time (in this case of figure). Simplifying, we can write $t_{tot} = t_B + N t_{sim}$ where t_B accounts for the one-time startup cost and $t_{sim} = t_F + t_T + t_S$ aggregates the time spent in running the CCN dynamics per-node. A linear regression yields to a bootstrap time of $t_B = 7\,\text{min}$ and $t_{sim} = 0.3\,\text{min/node}$ as for the CPU time needed to simulate 2 hour of YouTube catalog. This simple model tells us that (letting aside RAM bottlenecks) about 2-days of CPU time are sufficient to simulate 2 hours of YouTube catalog over a 10,000 nodes network with ccn$\mathcal{S}$im.

[9] In our case, we use LRU replacement and fill caches with content proportionally to its popularity; however, due to filtering effect, the most popular content can be evicted from some caches, and in case of non-LRU replacement, naive warm-up phases can result in an overall lengthen of the simulation duration. For such reasons, hot start is *disabled* by default in ccn$\mathcal{S}$im.

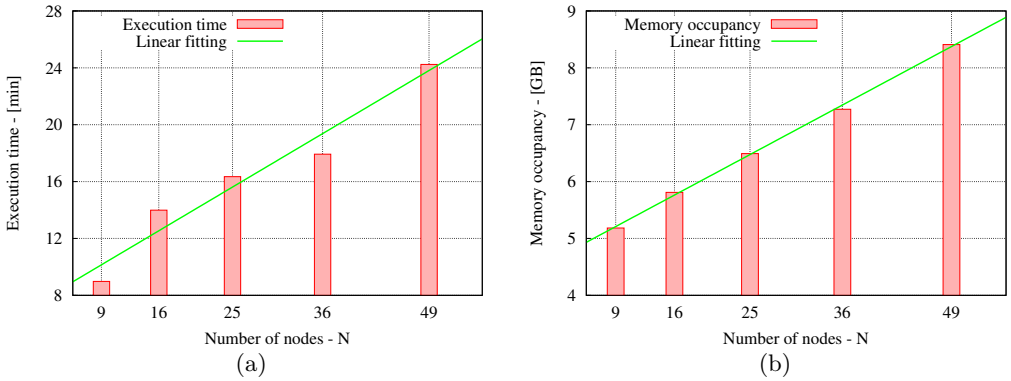


Figure 8.6: ccn$\mathcal{S}$im (a) Execution time and (b) Memory occupancy.

Fig. 8.6(b) depicts the memory occupancy for each individual simulation. As we can see, the largest simulated 50-nodes network requires about 8GB of RAM: thus, despite we cannot parallelize a single simulation with PDES, we can in principle run several single-core simulations in parallel[10]. To estimate RAM requirements, consider that the catalog stores several useful information (e.g., the repositories who own the permanent copies of each content), and that furthermore every CCN node implements a CS and PIT data structures[11]. Fig. 8.6(b) plots the memory consumption as a function of the network size. A linear regression over Fig. 8.6(b) data yields $m_{tot} = m_C + Nm_N$ where $m_C = 4.5\,\text{GB}$ is the baseline memory occupancy for handling a YouTube-like catalog (and all its associated fields), and $m_N = 80\,\text{MB/node}$ accounts for CCN nodes memory requirement (to store FIB, PIT and 10^6 chunks-long CS data structures).

Summarizing, we find that the major factor affecting ccn$\mathcal{S}$im scalability, either in terms of memory and latency, is not related to the size of the network. Rather, the main factor is the size of the catalog, especially for what concerns RAM memory occupancy, that we have seen to represent the crudest bottleneck. In future work, we aim at reducing these catalog memory requirement by implementing a memory efficient approximation of Zipf-like functions. For instance, a sizeable improvement could be achieved by faithfully representing only the 99th bulk of the catalog with a Zipf, while serving the remaining 1th percentile of the requests for rare content with a fictitious incremental ID. For the sake of the example, on a 10^8 catalog with $\alpha = 1$, this trick would save about 20% of the memory

[10]Consider however that this kind of parallelism may tradeoff with the scale of the scenario. Indeed, the catalog already represents a significant memory footprint (a 10^8 array of 64bit integers requires about 1GB of RAM), and clearly memory requirement grows with the catalog size, the number of chunks per file, the size of individual CCN caches and the size of the CCN network.

[11]The FIB does not directly impact RAM requirement, as it could be responsible for significant RAM footprints only for huge networks.

(reducing the catalog footprint from 4.G GB to 3.7 GB). We are currently investigating the use of variable lenght prefixes (e.g., use fewer bits for the most frequently accessed content) to represent content names in PIT/CS/FIB structures, that could bring further advantages in terms of the memory footprint.

8.4 Conclusions

This chapter surveys open source software tools for Information Centric Networking. On the one hand, our census testifies that, overall, a rather large number of simulators, emulators and real prototypes are available. On the other hand, we also see that generally only prototype implementations of specific ICN architectures are available.

Exceptions to this are represented by the Publish Subscribe Internet Routing Paradigm (PSIRP) and Content Centric Network (CCN) architectures. As for PSIRP, an Omnet++ simulator is available along with a couple of prototype implementations. As for CCN, not only a prototype, but also several other simulators are available. These tools correspond to different compromise between the realism vs complexity design space, and are indeed all necessary as they cover different aspects and scales of a complete CCN performance picture. Moreover, the availability of both custom simulators and of tools based on standard frameworks (such as ns3 and Omnet++) can further lower the startup cost (especially for users having previous experience with the ns3 or Omnet++ frameworks).

We next presented and benchmarked ccn$\mathcal{S}$im, one of the available CCN simulators that is based on Omnet++ framework and that especially targets caching and strategy layer performance, and whose extreme scalability is the joint result of key design choice, as well as a careful engineering (i.e., profiling and code refactoring). Briefly, our ccn$\mathcal{S}$im benchmarking shows that (i) a common off-the-shelf PC equipped with 8GB of RAM memory is able to simulate 2-hours of a 50-nodes CCN network, where each nodes is equipped with 10 GB caches, serving a 1 PB catalog in about 20 min CPU time; (ii) scalability issues are mainly tied to memory consumption, tied to catalog size more than content store of network nodes; (iii) parallel execution of ccn$\mathcal{S}$im is not a viable solution, as the major part of the additional CPU power is wasted in the synchronization among processes.

Overall, our census shows that few exceptions, ICN lacks of simulation tools that are commonly used to assist the algorithm design tasks of *each ICN architecture in isolation*. Moreover, the ICN domain as a whole misses so far any tool that could allow a *comparative evaluation of ICN architectures* – which will likely need more attention in the coming years.

Chapter 9

Conclusions

In this thesis we have studied, analyzed, modeled, and improved forwarding strategies for host and content centric networking. In this conclusive part, we summarize the best findings of this thesis, giving an overview about future and ongoing works.

9.1 Host Centric Networking

HCN philosophy replaces the thin waist of the communication hourglass with the concept of host. The overall idea is to directly query for the host flat identifier rather than hierarchical IP addresses. DHT-based approaches are widely used, nonetheless the underlying routing is based on traditional routing algorithms (usually, OSPF). Thus, in this field we consider the issue of multipath routing within the ISP boundaries (Part I), aim at finding an efficient replacement. The main findings are summarized below:

APL **analysis and evaluation** We propose APL (Ch. 3), a multipath algorithm suitable for interior AS boundaries. First we analyze APL 's costs and benefits, showing the message vs optimality trade-off: by slightly increasing the messages spread over the network, we can gather optimal disjoint end-to-end paths. The only knob for tuning the aforementioned trade-off is β, a system parameter representing the back-off base of the adaptive probabilistic procedure on which APL is based. However, β is easy to tune, and APL performance show to be robust against wrong β settings.

APLASIA **evaluation** We finally plug APL within APLASIA(Ch. 4). In particular, APLASIA exploits APL paths by the means of an autoforwarding data plane. First, we extend our previous statical APL analysis by providing an APL time complexity model: we prove APL termination, and evaluate its duration. Finally, we evaluate APLASIA autoforwarding plane by the means of Click experiments deployed on small scale testbeds. The coupling of APL with an autoforwarding data plane produces a fast convergent routing algorithm, jointly with a fast data plane which exploits the

paths provided by APL . The performance increase trade-offs with a slight increase in the communication cost of the control plane algorithm, which remains however similar to that of a simpler OSPF routing.

9.1.1 Future work

Even though HCN has been widely studied in terms of host-to-host forwarding, few works tackle the problem from an interior routing perspective. In this sense, APLASIA represents a novel architecture that can be easily extended. We list below the major future and ongoing directions in improving APLASIA's design. The final goal is to produce a valid (i.e., multipath, fast, automatic, self-configuring) OSPF substitute, suitable for the higher level HCN architectures.

Amount of control plane state Having an approach for precisely estimate the number of overlapping control plane message, we could reduce the control plane state to a $O(1)$ 8-bit counters, thus simplifying APLASIA's control plane. Indeed, overlapping advertisements represents the issue solved by retaining more than one single counter. By enhancing the analytical APL models, it is possible to calculate the probability that two different advertisements overlap, thus fixing a constant set of them without deteriorating performance.

Failure Resolution and Recovery As mentioned within Sec. 4.6, an important aspect, neglected within APLASIA's description, is represented by the failure resolution and recovery mechanisms. Upon a link failure, two different and parallel procedures have to happen: path caches must be updated , and new routes have to be recalculated (failure recovery). A failure mechanism should accurately trade-off between the complexity of the core routers and the recovery speed. Indeed, core nodes which detect the failure may just redirect data on the other path they have in cache. Edge nodes, instead, have to roll back the data up to the first core router able to switch the path and run another advertisement round.

9.2 Content Centric Networking

Information Centric Networking (ICN) moves a step further HCN, indicating pointing contents as the new thin waist of the hourglass model. Thus, instead of caching routes toward hosts, nodes can cache directly contents. This means that the thorough caching theory can be applied, but at a larger extent. Indeed, instead of single caches, ICN philosophy deals with (general) network of caches, and Network Caching Algorithms (NCA).

NCA assessment We assess performance of caching algorithms (Ch. 6), ranking the different factors affecting NCA behaviour. We consider large scale scenarios, to be sure of gathering a significant assessment. In particular, we show that the most influential

variable turn to be the popularity distribution: for instance, slightly changing the shaping factor of the Zipf distribution, may completely change algorithms performance. Then comes the algorithms strategies $\mathcal{F}$, $\mathcal{D}$, $\mathcal{R}$. Finally, the topology has minor impact: even trying to exploit its properties (e.g., betweenness centrality, connectivity, and hence forth) slightly improves the overall performance. For fulfilling this assessment, we designed, developed and optimized ccn$\mathcal{S}$im, whose implementation and core design is discussed in Ch. 8. We distributed ccn$\mathcal{S}$im as open source software, available at http://www.enst.fr/~drossi/ccnsim.

Forwarding strategies We particularly focus on forwarding strategies $\mathcal{F}$ (Ch. 7). We discuss exploitative strategies (i.e., strategies for which the paths are predetermined by some unspecified routing algorithm), showing that in some cases it can even worsens performance, especially when multiple paths are considered. Thus, we focus on exploratory forwarding strategies (flooding based). First, we investigate the performance of an ideal technique, the ideal Nearest Replica Routing (iNRR). iNRR nodes query an oracle for each content, retrieving its exact location. We show that iNRR reaches the best performance only if coupled with meta caching policies $\mathcal{D}$. Based on this insight, we develop two different Nearest Replica Routing (NRR) approaches, both based on TTL-scoped flooding:

- NRR' floods real interests, triggering the replacement and decision mechanisms at each node they traverse.
- NRR" floods first meta interests over the network, looking for the exact position of the content within the network. Then, the real interest is sent for downloading the content just discovered.

NRR" greatly outperforms NRR': this is because flooding real interests forces higher pollution within the caches. NRR", although slightly slower than NRR" shows to behave remarkably better. Roughly speaking, with small TTL values we fairly approach ideal performance.

9.2.1 Future work

This part has been mostly focused on the understanding of the dependencies between the different strategies $\langle \mathcal{F}, \mathcal{D}, \mathcal{R} \rangle$ of an NCA's triplet. In particular, we focus on a sort of quest for determining the best strategy coupling (e.g., ⟨NRR",LCD,LRU⟩).

The real issue in this analysis has been the absence of a ground truth: a best theoretical bound against which comparing our results. For instance, in the case of a single cache (with a static popularity law), the ground truth is well approximated by the LFU replacement. Thus, it is quite simple comparing each replacement strategy versus LFU for gathering the best one. This is not the case for NCAs, as the choice of which element to cache does

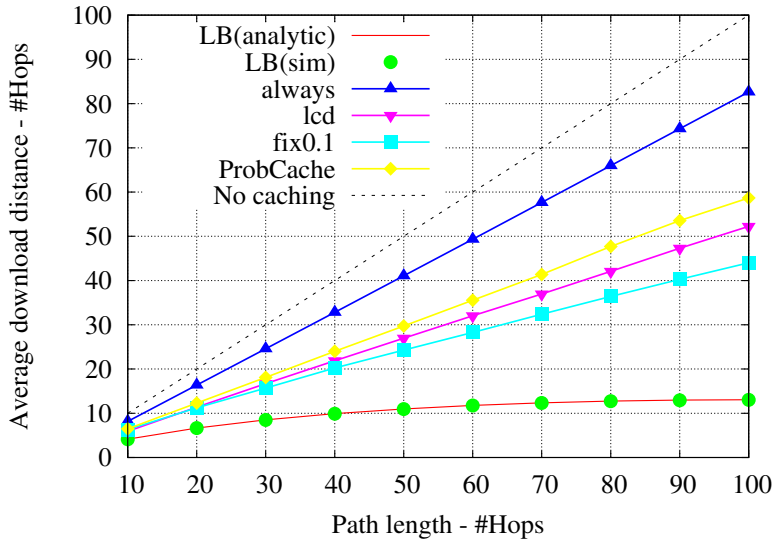


Figure 9.1: Optimal solution vs meta-caching strategies within a bus network of increasing size.

not solely depend by the popularity law. In the following, we summarize our ongoing and future work on the field of network caching algorithms.

Optimal forwarding In Fig. 9.1, we consider a bus of n caches: users are connected at one end and the single repository at the other end of the bus. In this case, the forwarding $\mathcal{F}$ is irrelevant. Fixing $\mathcal{R}$ to LRU, $\mathcal{D}$ represents the only free element to change of the $\langle \mathcal{F}, \mathcal{D}, \mathcal{R} \rangle$ triplet. As we can observe, on such a scenario the best strategy turns to be the FIX($\frac{10}{100}$) meta-caching, which is the strategy which gets closest to the lower bound. The analytic lower bound is fairly simple to estimate. The key insight is that, in the above scenario we should store popular content closer to the users and the unpopular one farther (possibly in the server). By supposing replicas placed in this way, and with some simple algebraic operations, we can calculate the lower bound of the average distance $\tilde{d} = z\, log \left[\frac{(\frac{n}{C})^n}{n!} \right]$, $z = (\sum p(i))$.

While the analytic lower bound results easy to determine in this particular case, we should try to generalize the analysis to more complex networks. By now, the only feasible solution comes from solving the huge optimization problem [155]. This latter turns to have Nn^2 variables, and resulting computationally prohibitive to solve on realistic scenarios.

Dynamic popularity From a wider system perspective, we consider through all the second part a fixed probability law, space and time independent. While some investiga-

tion about space dependent popularity has been already inspected by our previous work [106], there is a lack of investigation about time dependent popularity. We remark that having a time dependent catalog may increase NCA performance: indeed, roughly speaking, this could result in a sort of smaller catalog, within a given time interval. Thus, understanding how the popularity of a given content changes (e.g.,identifying a well-defined analytical model) is crucial for NCA performance and surely represents one of the right direction to investigate.

Increasing cache size Instead of grasping complex $\langle \mathcal{F}, \mathcal{D}, \mathcal{R} \rangle$ strategies, we can apply a really simpler approach: increasing the cache space. Usually, this sort of naïve solution incurs in the technological limits [47] of the DRAM and SRAM chips, employed for implementing caches (and indexes) in CCN/ICN routers (see also Sec. 6.1). Nonetheless we know that, in particular for the CCN architecture, objects can be subdivided in chunks (see Sec. 5.1) to be sequentially downloaded. The basic idea on which we are currently working is to leverage on the correlation among subsequent requests for the same content, thus implementing a fast and larger *two layer cache router*. Its basic functioning is as follows: a first slow cache (first layer) stores the whole file, while a faster cache (second layer) stores only some chunks of the same object. When an interest arrives (and if it is actually present in the slow cache), the fast cache returns back the first chunks of the content. Meanwhile, the file is transferred into the fast memory to be seamlessly downloaded by the next requests. In this way, users actually see a huge and fast cache, suffering delays at most for the very first chunk of the object – in the case the content is in the slow memory, but not in the fast one.

Appendix A

Publications

Journals

1. G. Rossini, D. Rossi, Evaluating CCN multi-path interest forwarding strategies . Elsevier Computer Communication, SI on Information Centric Networking, 2013.

2. Giuseppe Rossini, Dario Rossi, Christophe Betoule, Remi Clavier and Gilles Thouenon, FIB Aplasia through Probabilistic Routing and Autoforwarding . Elsevier Computer Networks (to appear), 2013.

3. G.Rossini, D.Rossi, Exploiting topology knowledge in Information Centric Networks: Guidelines and challenges. MMC E-LETTER, 2013

Conferences

1. G. Rossini, D. Rossi, M. Garetto, E. Leonardi, Multi-Terabyte and Multi-Gbps Information Centric Routers. In IEEE INFOCOM 2014 (To Appear)

2. R. Chiocchetti, D. Perino, G. Carofiglio, D. Rossi, G. Rossini, INFORM: a Dynamic Interest Forwarding Mechanism for Information Centric Networking . In ACM SIGCOMM Worskhop on Information Centric Networking

3. Chiocchetti, Raffaele, Rossi, Dario and Rossini, Giuseppe, ccnSim: an Highly Scalable CCN Simulator . In IEEE International Conference on Communications (ICC), june 2013.

4. G. Rossini, D. Rossi, A dive into the caching performance of Content Centric Networking . In IEEE 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD'12), 2012.

5. G. Rossini, A Tutorial on ccnSim, an Highly Scalable CCN simulator, Talk at RESCOM 2013

6. Raffaele Chiocchetti, Dario Rossi, Giuseppe Rossini, Giovanna Carofiglio and Diego Perino„ Exploit the known or explore the unknown: Hamlet-like doubts in ICN. In ACM SIGCOMM, ICN Workshop, 2012.

7. G. Rossini and D. Rossi, Large scale simulation of CCN networks. In Algotel 2012 , La Grande Motte, France, May 2012.

8. C. Betoule, T. Bonald, R. Clavier, D. Rossi, G. Rossini and G. Thouenon,Adaptive Probabilistic Flooding for Multi-path Routing . In IFIP NTMS, Best paper award, 2012.

9. D. Rossi and G. Rossini, On sizing CCN content stores by exploiting topological information . In IEEE INFOCOM, NOMEN Worshop, , Orlando, FL, March 25-30 2012.

Patents

1. D. Perino, G. Carofiglio, D. Rossi and G. Rossini, Dynamic Interest Forwarding Mechanism for Information Centric Networking . Patent EP 13 306 124.2, filed 5/8/2013.

2. D. Perino, G. Carofiglio, R. Chiocchetti, D. Rossi and G. Rossini, Device and method for organizing forwarding information in nodes of a content centric networking . Patent EPO13161714.4 - 1856, filed 28/03/2013.

Bibliography

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *Communications Magazine, IEEE*, vol. 50, no. 7, pp. 26 –36, july 2012.

[2] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. M. Maggs, K. Ng, V. Sekar, and S. Shenker, "Less pain, most of the gain: Incrementally deployable icn," in *ACM SIGCOMM*, 2013.

[3] G. Carofiglio, M. Gallo, and L. Muscariello, "Bandwidth and Storage Sharing Performance in Information Centric Networking," in *ACM SIGCOMM, ICN Worskhop*, 2011, pp. 1–6.

[4] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Modeling Data Transfer in Content-Centric Networking," in *ITC*, 2011.

[5] P. R. Jelenković, "Asymptotic approximation of the move-to-front search cost distribution and least-recently used caching fault probabilities," *The Annals of Applied Probability*, vol. 9, no. 2, 1999.

[6] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for lru cache performance," in *ITC*, 2012.

[7] H. Che, Z. Wang, and Y. Tung, "Analysis and design of hierarchical web caching systems," in *IEEE INFOCOM*, 2001, pp. 1416–1424.

[8] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate Models for General Cache Networks," *IEEE INFOCOM*, pp. 1–9, 2010.

[9] G. Rossini and D. Rossi, "Evaluating ccn multi-path interest forwarding strategies," *Computer Communications*, vol. 36, no. 7, 2013.

[10] R. Chiocchetti, D. Rossi, G. Rossini, G. Carofiglio, and D. Perino, "Exploit the known or explore the unknown?: hamlet-like doubts in icn," in *ACM SIGCOMM ICN*, Helsinki, Finland, August 2012.

[11] R. Chiocchetti, D. Perino, G. Carofiglio, D. Rossi, and G. Rossini, "INFORM: a dynamic interest forwarding mechanism for information centric networking," in *ACM SIGCOMM, ICN*, 2013.

[12] S. Eum, K. Nakauchi, M. Murata, Y. Shoji, and N. Nishinaga, "CATT: potential based routing with content caching for icn," in *ACM SIGCOMM , ICN*, 2012.

[13] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A case for stateful forwarding plane," *Computer Communications*, vol. 36, no. 7, 2013.

[14] E. J. Rosensweig and J. Kurose, "Breadcrumbs: Efficient, Best-Effort Content Location in Cache Networks," *IEEE INFOCOM*, 2009.

[15] N. Laoutaris, S. Syntila, and I. Stavrakakis, "Meta Algorithms for Hierarchical Web Caches," in *IEEE ICPCC*, 2004.

[16] N. Laoutaris, H. Che, and I. Stavrakakis, "The LCD interconnection of LRU caches and its analysis," *Performance Evaluation*, vol. 63, no. 7, 2006.

[17] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *ACM ICN*, 2012.

[18] K. Cho, M. Lee, K. Park, T. Kwon, Y. Choi, and S. Pack, "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," in *IEEE NOMEN*, 2012.

[19] W. Chai, D. He, I. Psaras, and G. Pavlou, "Cache "less for more" in information-centric networks," in *IFIP NETWORKING*, 2012, vol. 7289, pp. 27–40.

[20] M. Handley, "Why the internet only just works," *BT Technology Journal*, vol. 24, no. 3, 2006.

[21] A. Passarella, "A survey on content-centric technologies for the current internet: CDN and P2P solutions," *Computer Communications*, vol. 35, no. 1, 2012.

[22] L. Popa, A. Ghodsi, and I. Stoica, "Http as the narrow waist of the future internet," in *HotNets*, 2010.

[23] A. M. Odlyzko, "Internet traffic growth: sources and implications," vol. 5247, 2003, pp. 1–15.

[24] B. Ford and J. Iyengar, "Breaking up the transport logjam," in *ACM HotNets, October*, 2008.

[25] X. Zhao, D. Pacella, and J. Schiller, "Routing scalability: An operator's view," *Selected Areas in Communications, IEEE Journal on*, vol. 28, no. 8, 2010.

[26] Cisco visual networking index: Forecast and methodology, 2009-2014.

[27] T. L. Rodeheffer, C. A. Thekkath, and D. C. Anderson, "Smartbridge: a scalable bridge architecture," in *ACM SIGCOMM*, August 2000.

[28] R. Perlman, "Rbridges: transparent routing," in *IEEE INFOCOM*, March 2004, pp. 1211 – 1218.

[29] S. Sharma, K. Gopalan, S. Nanda, and T. Chiueh, "Viking: a multi-spanning-tree ethernet architecture for metropolitan area and cluster networks," in *IEEE INFOCOM*, March 2004.

[30] C. Kim, M. Caesar, and J. Rexford, "Floodless in SEATTLE: a scalable Ethernet architecture for large enterprises," in *ACM SIGCOMM*, August 2008, pp. 3–14.

[31] H. T. Kaur, S. Kalyanaraman, A. Weiss, S. Kanwar, and A. Gandhi, "Bananas: an evolutionary framework for explicit and multipath routing in the internet," in *ACM FDNA*, August 2003, pp. 277–288.

[32] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, and I. Stoica, "ROFL: routing on flat labels," in *ACM SIGCOMM*, August 2006, pp. 363–374.

[33] A. Singla, P. B. Godfrey, K. Fall, and S. Iannaccone, G. Ratnasamy, "Scalable routing on flat names," in *ACM CoNEXT*, November 2010.

[34] N. R. Mysore, A. Pamboris, N. Farrington, P. Huang, N. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM*, August 2009.

[35] D. Eppstein, "Finding the k shortest paths," *IEEE FOCS*, pp. 154–165, November 1994.

[36] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. Mogul, "Spain: Cots data-center ethernet for multipathing over arbitrary topologies," in *USENIX NSDI*, April 2010, pp. 18–34.

[37] R. Ogier, V. Rutenburg, and N. Shacham, "Distributed algorithms for computing shortest pairs of disjoint paths," *IEEE Information Theory*, vol. 39, no. 2, pp. 443 –455, Mar. 1993.

[38] A. Varga, "Omnet++ website," http://www.omnetpp.org, 2010.

[39] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second igp convergence in large ip networks," *ACM SIGCOMM CCR*, vol. 35, pp. 35–44, July 2005.

[40] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, pp. 263–297, August 2000.

[41] V. Jacobson, D. Smetters, N. Briggs, J. Thornton, M. Plass, and R. Braynard, "Networking Named Content," in *ACM CoNEXT*, August 2009.

[42] C. Yi, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "Adaptive forwarding in named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 3, 2012.

[43] R. Fonseca, M. Crovella, and B. Abrahao, "On the Intrinsic Locality Properties of Web Reference Streams," in *IEEE INFOCOM*, 2003.

[44] S. Arianfar and P. Nikander, "Packet-level Caching for Information-centric Networking," in *ReArch Workshop*, 2010.

[45] M. Gallo, B. Kauffmann, L. Muscariello, A. Simonian, and C. Tanguy, "Performance evaluation of the random replacement policy for networks of caches," in *ACM SIGMETRICS*, 2012.

[46] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," in *ACM IMC*, 2007, pp. 15–28.

[47] D. Perino and M. Varvello, "A reality check for content centric networking," in *ACM ICN*, 2011.

[48] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," in *ACM IMC*, 2007.

[49] D. Rossi, C. Testa, and S. Valenti, "Yes, we ledbat: Playing with the new bittorrent congestion control algorithm," in *PAM*, 2010.

[50] Y. Cui, P. Wu, M. Xu, J. Wu, Y. Lee, A. Durand, and C. Metz, "4over6: network layer virtualization for ipv4-ipv6 coexistence," *Network, IEEE*, vol. 26, no. 5, 2012.

[51] P. H. Salus, *Casting the Net: From ARPANET to Internet and Beyond...* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[52] C. S. Carr, S. D. Crocker, and V. G. Cerf, "Host-host communication protocol in the arpa network," in *ACM AFIPS*, 1970.

[53] D. Clark, "The design philosophy of the darpa internet protocols," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, 1988.

[54] Cisco visual networking index: Forecast and methodology, 2009-2014.

[55] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *ACM SIGCOMM*, 2001.

[56] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM*, 1988.

[57] M. Thorup and U. Zwick, "Compact routing schemes," in *ACM SPAA*, 2001.

[58] R. Sanchez, L. Raptis, and K. Vaxevanakis, "Ethernet as a carrier grade technology: developments and innovations," *IEEE Communications Magazine*, vol. 46, pp. 88–94, 2008.

[59] "Ieee standard 802.1ah - provider backbone bridges (pbb)," June 2008.

[60] "Ieee 802.1qay - provider backbone bridge traffic engineering (pbb-te)," Jan. 2009.

[61] E. Mannie, "Generalized multi-protocol label switching (gmpls) architecture," in *IETF RFC3945,.*

[62] J. J. Garcia-Lunes-Aceves, "Loop-free routing using diffusing computations," *IEEE/ACM Trans. Netw.*, vol. 1, February 1993.

[63] W. Zaumen and J. Garcia-Luna-Aceves, "Loop-free multipath routing using generalized diffusing computations," in *IEEE INFOCOM*, March 1998, pp. 1408–1417.

[64] M. J. Blesa and C. Blum, "Ant colony optimization for the maximum edge-disjoint paths problem," in *Applications of Evolutionary Computing*, vol. 3005, 2004, pp. 160,169.

[65] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "Mate: Mpls adaptive traffic engineering," in *ACM SIGCOMM*, vol. 3, 2001, pp. 1300 –1309.

[66] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," RFC 2992 (Informational), Internet Engineering Task Force, Nov. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2992.txt

[67] J. Hershberger, M. Maxel, and S. Suri, "Finding the k shortest simple paths: A new algorithm and its implementation," *ACM Trans. Algorithms*, vol. 3, November 2007.

[68] P. Merindol, J. Pansiot, and S. Cateloin, "Low complexity link state multipath routing," in *IEEE INFOCOM Workshop*, april 2009, pp. 1 –6.

[69] N. Lin and Z. Shao, "Improved ant colony algorithm for multipath routing algorithm research," in *IEEE IPTC*, oct. 2010, pp. 651 –655.

[70] D. Johnson, Y. Hu, and D. Maltz, "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4," IETF RFC 4728, Feb. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4728.txt

[71] F. Solano, T. Stidsen, R. Fabregat, and J. Marzo, "Label space reduction in mpls networks: How much can a single stacked label do?" *IEEE/ACM Transactions on Networking,*, vol. 16, no. 6, 2008.

[72] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica, "Pathlet routing," in *ACM SIGCOMM*, August 2009, pp. 111–122.

[73] M. J. Kim, D. H. Lee, and Y. I. Eom, "Enhanced non-disjoint multi-path source routing protocol for wireless ad-hoc networks," in *ACM ICCSA*, 2007, pp. 1187–1196.

[74] A. Myers, T. Eugence, and H. Zhang, "Rethinking the service model: Scaling ethernet to a million nodes," in *ACM Hotnet*, August 2004.

[75] C. Kim, M. Caesar, A. Gerber, and J. Rexford, "Revisiting route caching: The world should be flat," in *ACM PAM*, April 2009, pp. 3–12.

[76] D. Johnson, Y. Hu, and D. Maltz, "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4," IETF RFC 4728, Feb. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4728.txt

[77] S. Nelakuditi and Z.-L. Zhang, "On selection of paths for multipath routing," in *IEEE IWQoS*, vol. 2092, 2001, pp. 170–184.

[78] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," in *ACM SIGCOMM*, August 2002, pp. 133–145.

[79] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, "Path splicing," in *ACM SIGCOMM*, August 2008, pp. 27–38.

[80] N. Feamster, D. G. Andersen, H. Balakrishnan, and M. F. Kaashoek, "Measuring the effects of internet path faults on reactive routing," *ACM SIGMETRICS*, pp. 126–137, June 2003.

[81] B. Bollobas, *Random Graphs*. Cambridge University Press, 2001.

[82] A. Raj and O. C. Ibe, "A survey of ip and multiprotocol label switching fast reroute schemes," *Computer Networks*, vol. 51, June 2007.

[83] J. Moy, "OSPF Version 2," RFC 2328 (Standard), Internet Engineering Task Force, apr 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2328.txt

[84] V. Jacobson, D. K. Smetters, N. H. Briggs, J. D. Thornton, M. F. Plass, and R. L. Braynard, "Networking Named Content," in *CoNEXT*, 2009.

[85] L. Wang, A. K. M. Hoque, C. Yi, A. Alyyan, and B. Zhang, "OSPFn: An OSPF based routing protocol for named data networking," Tech. Rep., March 2012.

[86] D. Perino, G. Carofiglio, R. Chiocchetti, D. Rossi, and G. Rossini, "Device and method for organizing forwarding information in nodes of a content centric networking," Patent EPO13 161 714.4/2013.

[87] S. Podlipnig and L. B. Osz, "A Survey of Web Cache Replacement Strategies," vol. 35, no. 4, pp. 374–398, 2003.

[88] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: Modeling, design and experimental results," *IEEE JSAC*, vol. 20, no. 7, pp. 1305–1314, 2002.

[89] T. Wong, G. Ganger, and J. Wilkes, "My cache or yours? making storage more exclusive," in *USENIX Annual Technical Conference*, 2002.

[90] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy, "On the scale and performance of cooperative Web proxy caching," in *ACM SOSP*, 1999, pp. 16–31.

[91] M. Fiore, F. Mininni, C. Casetti, and C.-F. Chiasserini, "To cache or not to cache?" in *IEEE INFOCOM*, 2009, pp. 235 –243.

[92] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou, "Modelling and Evaluation of CCN-Caching Trees," *IFIP Networking*, 2011.

[93] J. Choi, J. Han, E. Cho, T. T. Kwon, and Y. Choi, "A Survey on Content-Oriented Networking for Efficient Content Delivery," *IEEE Communications Magazine*, pp. 121–127, 2011.

[94] K. Katsaros, G. Xylomenos, and G. Polyzos, "MultiCache : An overlay architecture for information-centric networking," *Computer Networks*, pp. 1–11, 2011.

[95] J. Ardelius, B. Gronvall, L. Westberg, and A. Arvidsson, "On the effects of caching in access aggregation networks," in *ACM ICN*, Helsinki, Finland, 2012.

[96] K. Park and V. Pai, "Scale and performance in the coblitz large-file distribution service," in *USENIX NSDI*, 2006.

[97] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *IEEE/ACM Transactions on Networking*, vol. 16, no. 6, 2008.

[98] M. Busari and C. Williamson, "ProWGen: a synthetic workload generation tool for simulation evaluation of Web proxy caches," *Computer Networks*, vol. 38, no. 6, 2002.

[99] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: findings and implications," in *ACM SIGMETRICS*, 2009, pp. 37–48.

[100] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," in *ACM SIGCOMM*, 2002.

[101] ccnSim homepage. http://www.infres.enst.fr/~drossi/ccnSim.

[102] A. Narayanan and D. Oran, "Content Routing using Internet Routing Protocols: Can it scale?" IETF-82, ICNRG BAR BOF, 2011, uRL: http://trac.tools.ietf.org/group/irtf/trac/wiki/icnrg.

[103] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Information-centric networking: seeing the forest for the trees," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. ACM, 2011, p. 1.

[104] A. Dan and D. Towsley, "An approximate analysis of the lru and fifo buffer replacement schemes," *ACM SIGMETRICS*, 1990.

[105] G. Rossini and D. Rossi, "ccnSim: an highly scalable ccn simulator," in *IEEE ICC*, 2013.

[106] ——, "A dive into the caching performance of content centric networking," in *IEEE CAMAD*, 2012.

[107] G. Rossini, D. Rossi, M. Garetto, and E. Leonardi, "Multi-terabyte and multi-gbps information centric routers," Tech. Rep., 2013.

[108] http://mobilityfirst.winlab.rutgers.edu/Prototype.html.

[109] NSF FIA MobilityFirst. http://mobilityfirst.winlab.rutgers.edu/.

[110] http://users.piuha.net/blackhawk/0.3/.

[111] FP7 Publish Subscribe Internet Rouring Paradigm (PSIRP). http://www.psirp.org/.

[112] FP7 PPURSUIT . http://www.fp7-pursuit.eu/.

[113] https://github.com/fp7-pursuit/blackadder.

[114] http://privatewww.essex.ac.uk/~nvasta/ICNSim.htm.

[115] TSB/EPSRC Personal and Social Communication Services for Health and Lifestyle Monitoring (PAL). http://palproject.org.uk.

[116] https://sourceforge.net/projects/netinf.

[117] FP7 Scalable and Adaptive Internet Solutions (SAIL). http://www.sail-project.eu/.

[118] http://code.google.com/p/opennetinf/.

[119] FP7 4WARD. http://www.4ward-project.eu/.

[120] http://www.ict-convergence.eu/wp-content/uploads/CPK.zip.

[121] FP7 CONVERGENCE. http://www.ict-convergence.eu/.

[122] http://netgroup.uniroma2.it/CONET/.

[123] Ccnx homepage. http://www.ccnx.org/.

[124] NSF Named Data Networking (NDN). http://www.named-data.org/.

[125] http://www-sop.inria.fr/members/Frederic.Urbani/ns3dceccnx/.

[126] Content-Oriented Networking: a New Experience for Content Transfer (CONNECT). http://anr-connect.org/.

[127] "Ns-3 based named data networking (ndn) simulator," http://ndnsim.net.

[128] http://code.google.com/p/ccnpl-sim/.

[129] FP7 COntent Mediator architecture for content-aware nETworks (COMET). http://www.comet-project.org.

[130] FP7 Content Aware Searching retrieval and sTreaming (COAST). http://www.coast-fp7.eu.

[131] http://swig.org/.

[132] The click modular router project. http://www.read.cs.ucla.edu/click/.

[133] Getting started with ns-3-click and nsclick. http://www.read.cs.ucla.edu/click/nsclick.

[134] K. Y. M. R. N. Vastardis, A. Bontozoglou, "Simulation Tools Enabling Research on Information-centric Networks," in *IEEE ICC 2012 Workshop on the Network of the Future (FutureNet V)*, June.

[135] S. F. et al., "Naming things with hashes."

[136] P. H.-B. et al., "The named information (ni) uri scheme: Optional features,."

[137] FP7 OpenFlow in Europe Linking Infrastructure and Application (OFELIA). http://http://www.fp7-ofelia.eu/.

[138] N. B.-M. A. Detti, S. Salsano.

[139] www.openflow.org/.

[140] S. S.-N. B.-M. A. D. L. Veltri, G. Morabito, "Supporting information-centric functionality in software defined networks," in *IEEE ICC Workshop on Software Defined Networks (SDN),*.

[141] A. T.-G. Jonas Eymann, Yunqi Luo, "OMNeT++ based Simulator for Content Centric Networking," in *CCNxCon'12*, September 2012.

[142] http://www.named-data.net/testbed.html.

[143] https://www.grid5000.fr/.

[144] http://planet-lab.org/.

[145] Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang, "ndnSIM: NDN simulator for NS-3."

[146] L. Z. Alex Afanasyev, Ilya Moiseenko, "ndnSIM: A ns-3 Based NDN Simulator," in *CCNxCon'12*, September 2012.

[147] http://www.inf.usi.ch/carzaniga/cbn/routing/index.html.

[148] A. Carzaniga, M. Rutherford, and A. Wolf, "A routing scheme for content-based networking," in *IEEE INFOCOM*, 2004.

[149] G. Carofiglio, M. Gallo, and L. Muscariello, "Joint hop-by-hop and receiver-driven interest control protocol for content-centric networks," in *ACM SIGCOMM Workshop on Information-centric networking (ICN)*, 2012.

[150] ——, "Icp: Design and evaluation of an interest control protocol for content-centric networking," in *IEEE INFOCOM Workshop of Emerging Design Choices in Name-Oriented Networking (NOMEN'2012)*, 2012.

[151] "Omnet++ homepage," http://www.omnetpp.org/.

[152] B. Stroustrup, *The C++ Programming Language*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[153] Boost homepage. http://www.boost.org/.

[154] R. L. Bagrodia and M. Takai, "Performance evaluation of conservative algorithms in parallel simulation languages," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, pp. 395–411, 2000.

[155] I. D. Baev and R. Rajaraman, "Approximation algorithms for data placement in arbitrary networks," in *ACM SODA*. Society for Industrial and Applied Mathematics, 2001.