UNIVERSITY OF THESSALY, GREECE

Replication Management and Cache Aware Routing in Information-Centric Networking

Doctor of Philosophy

in

Electrical & Computer Engineering

by

Vasilis Sourlas

July 2013

The Dissertation of Vasilis Sourlas is approved by:

Committee Chairperson

Electrical & Computer Engineering,
University of Thessaly, Greece

ACKNOWLEDGMENTS

This thesis represents a culmination of research work that has taken place at the Department of Electrical and Computer Engineering, University of Thessaly, Greece.

I would like to express my gratitude to all those who made this dissertation possible, and first of all to my supervisor, Professor Leandros Tassiulas for giving me an opportunity to work in such a joyful and inspiring research unit. Without his guidance and constructive criticism this doctoral study would not have been possible. He also gave me the chance to participate in many interesting European Union and National research projects; an experience that broadened my research view and played a key role in the completion of my PhD. I would also like to thank the other members of the dissertation committee for their contribution and the examination committee for reading this thesis.

I would like to thank my friends and lab-mates, who provided help and made the long journey a lot more enjoyable and memorable. Particularly, I would like to thank Dr. Paris Flegkas, Dr. Georgios Paschos and Lazaros Gkatzikis for their help and fruitful cooperation in many aspects of my research. All those numerous coffee breaks were really refreshing for the completion of my PhD. I would also like to thank my friend Dr. Konstantinos Manousakis which was the first person that I talked in my first day at the university and we still share our problems and agonies for our academic future.

Finally, I would also like to thank my family for their love and support. There are no words that can express my gratitude and appreciation for all they have done for me. Last but not least I want to thank Eleni Eliadou for her support all those years and her unlimited patience. The least I can do in recognition to their love and support is to dedicate this thesis to them.

Finally, I would like to acknowledge the support of the Heracleitus research project co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) Research Funding Program: Heracleitus II - Investing in knowledge society through the European Social Fund.

European Union
European Social Fund
OPERATIONAL PROGRAMME
EDUCATION AND LIFELONG LEARNING
investing in knowledge society
MINISTRY OF EDUCATION & RELIGIOUS AFFAIRS
MANAGING AUTHORITY
NSRF
2007-2013
programme for development
EUROPEAN SOCIAL FUND
Co-financed by Greece and the European Union

Dedicated to my family and Eleni Eliadou.

ABSTRACT OF THE DISSERTATION

Replication Management and Cache Aware Routing in Information-Centric Networking

by

Vasilis Sourlas

Doctor of Philosophy, Post Graduate Program in Electrical and Computer Engineering,
University of Thessaly, Greece.
July 2013
Prof. Leandros Tassiulas, Chairperson

Content distribution in the Internet places content providers in a dominant position, with delivery happening directly between two end-points, that is, from content providers to consumers. This model is driven by the underlying Internet routing paradigm, namely that of transferring datagrams from one routable endpoint to another. This location-centric model limits the ability to fully utilize resources that are available along the route from the provider to the consumer(s), such as storage (e.g., for caching) or computing (e.g., for re-encoding). Information-Centrism has been proposed as a paradigm shift from the host-to-host Internet to a host-to-content one, or in other words from an end-to-end communication system to a native distribution network. This trend has attracted the attention of the research community, which has argued that content, instead of end-points, must be at the center stage of attention. These efforts promise, among other things, more flexibility in adapting to new services, efficiency improvements on lower layers, and native multicast support.

Given this emergence of information-centric solutions, the relevant management needs in terms of performance have not been adequately addressed, yet they are absolutely essential for relevant network operations and crucial for the information-centric approaches to succeed. Performance management and traffic engineering approaches are also required to control routing, to configure the logic for replacement policies in cache stores and to control decisions where to cache, for instance. Therefore, there is an urgent need to manage information-centric resources and in fact to constitute their missing management and control plane which is essential for their success as clean-slate technologies. In this thesis we aim to provide solutions to crucial problems that remain, such as the management of information-centric approaches which has not yet been addressed, focusing on the key aspect of route and cache management.

Our main goal is to investigate and develop key network management functions for information-centric approaches related to route and cache management. Particularly, we have developed mechanisms that manage the routing processes by influencing the forwarding of interest/subscription packets and make caching decisions about where and which item to cache as well as influence the cache replacement policies. The aim is to improve the operations and overall utility of the network through extensive and novel usage of caching as an inherent architectural function. Both opportunistic in-network and service-specific managed caching (CDN-like replication) is of interest in addressing this challenge. We have also devised a set of caching solutions, evaluated through proper analytical models, that not only dramatically improve the overall utility but also demonstrate the further potential of the architectural paradigm of information-centrism.

In more details, we initially describe a three phase framework as a contribution to the problem of information replication. The objective of the proposed framework is to minimize the total traffic load in the network subject to installing a predefined number of replication devices, and given that each device has storage limitations. Particularly, we present and evaluate a new algorithm for the selection of the location in the network to install replication devices. We also propose two alternative off-line mechanisms for the assignment of the replicas of each information item among the selected replications devices. We also propose a distributed cache management architecture that dynamically (re-)assigns information items to caches in order to minimize the overall network traffic cost imposed by the user requests. In particular, we derive four distributed on-line intra-domain cache management algorithms, categorize them according to the level of cooperation needed and compare them in terms of performance, complexity, message overhead and convergence time. We derive also a lower bound of the overall network traffic cost for a certain class of network topologies and show that the proposed cache management algorithms perform closely to the derived lower bound.

We also present and decompose an in-network opportunistic caching framework in a set of basic policies, present at each set the most known and propose an information-centric policy at each one of them. We further propose a stochastic model that captures the dynamics of the newly proposed policies and describe a prototype implementation of the proposed opportunistic caching mechanism which is also evaluated through Planetlab experiments. Moreover, we present an enhancement of the opportunistic caching mechanism to enable client mobility. Finally, we propose a new cache aware intra-domain routing scheme that dynamically computes the routes followed by each interest/ subscription for each item and from each node in the network. Particularly, we present a dynamic programming (DP) approach for the computation of the minimum transportation cost paths based on the observed item request patterns in order to minimize the overall transportation cost imposed by the user requests.

ΠΕΡΙΛΗΨΗ ΤΗΣ ΔΙΔΑΚΤΟΡΙΚΗΣ ΔΙΑΤΡΙΒΗΣ

"Διαχείριση Αντιγράφων και Δρομολόγησης σε Δίκτυα που Οργανώνονται με βάση το Περιεχόμενο"

Βασίλειος Σούρλας

Η διαδικασία διανομής περιεχομένου στο Διαδίκτυο τοποθετεί τους παρόχους περιεχομένου σε δεσπόζουσα θέση, με την παράδοση του περιεχομένου να συμβαίνει μεταξύ δύο τελικών σημείων, δηλαδή, από τους παρόχους περιεχομένου στους καταναλωτές. Αυτό το μοντέλο οδηγείται από το υποκείμενο παράδειγμα δρομολόγησης του Διαδικτύου, δηλαδή της μεταφοράς πακέτων δεδομένων από το ένα τελικό σημείο στο άλλο. Αυτό το βασισμένο στην περιοχή (location-centric) μοντέλο περιορίζει τη δυνατότητα να αξιοποιηθούν πλήρως οι πόροι που είναι διαθέσιμοι κατά μήκος της διαδρομής από τον πάροχο στον καταναλωτή, όπως για παράδειγμα η προσωρινή δυνατότητα αποθήκευσης (caching). Τα δίκτυα που οργανώνονται με βάση το περιεχόμενο (Information-Centric Networks, ICN) έχουν προταθεί ως μια εναλλακτική της host-to-host επικοινωνίας σε μια host-to-content μορφή επικοινωνίας, ή με άλλα λόγια από την άκρο σε άκρο μορφή επικοινωνίας σε έναν εγγενή δίκτυο διανομής περιεχομένου. Η τάση αυτή έχει προσελκύσει την προσοχή της ερευνητικής κοινότητα, η οποία υποστηρίζει ότι το περιεχόμενο, αντί των τελικών σημείων, πρέπει να είναι στο επίκεντρο της προσοχής. Οι προσπάθειες αυτές υπόσχονται, μεταξύ άλλων, μεγαλύτερη ευελιξία για την προσαρμογή στις νέες υπηρεσίες, βελτίωση της απόδοσης στα κατώτερα στρώματα του δικτύου, και εγγενή υποστήριξη πολλαπλής διανομής (multicast).

Με δεδομένη την εμφάνιση ICN λύσεων, οι σχετικές ανάγκες διαχείρισης όσον αφορά τις επιδόσεις δεν έχουν επαρκώς οριστεί, παρά το γεγονός ότι είναι απολύτως απαραίτητες για την σχετική λειτουργία του δικτύου και ζωτικής σημασίας για την επιτυχία των ICN προσεγγίσεων. Η έννοια της ροής, για παράδειγμα, είναι εντελώς διαφορετική στα ICN δίκτυα σε σύγκριση με το IP δίκτυο και θεμελιώδης έρευνα είναι αναγκαία, ώστε να γνωρίζουμε πώς χρησιμοποιείται το δίκτυο. Οι διάφορες προσεγγίσεις για την διαχείριση των επιδόσεων και της κυκλοφορίας οφείλουν για παράδειγμα να ελέγξουν τη δρομολόγηση, να ρυθμίσουν τη λογική των πολιτικών αντικατάστασης στις προσωρινές μονάδες αποθήκευσης και να ελέγξουν που αποθηκεύεται κάθε στοιχείο πληροφορίας. Ως εκ τούτου, υπάρχει επείγουσα ανάγκη για τη διαχείριση των πόρων των ICN δικτύων και στην πραγματικότητα να σχηματιστεί το επίπεδο διαχείρισης και το επίπεδο ελέγχου τα οποία είναι απαραίτητη για την επιτυχία του ως μια νέα μορφή τεχνολογίας. Η παρούσα εργασία στοχεύει να δώσει λύσεις σε κρίσιμα προβλήματα που παραμένουν, όπως η διαχείριση των ICN δικτύων η οποία δεν έχει ακόμη αντιμετωπιστεί, με επίκεντρο τα βασικά χαρακτηριστικά της διαχείρισης της δρομολόγησης και των τεχνικών προσωρινής αποθήκευσης.

Ο κύριος στόχος μας είναι να διερευνήσουμε και να αναπτύξουμε βασικές λειτουργίες διαχείρισης των ICN δικτύων που σχετίζονται με τη δρομολόγηση και τη διαχείριση της προσωρινής αποθήκευσης. Συγκεκριμένα, έχουμε αναπτύξει μηχανισμούς που διαχειρίζονται τις διαδικασίες δρομολόγησης επηρεάζοντας τη διαβίβαση των interest/subscription πακέτων και παίρνουν αποφάσεις για το πού και ποιο κομμάτι πληροφορίας θα αποθηκευτεί στην προσωρινή μνήμη (cache), ενώ επίσης επηρεάζουν τις πολιτικές αντικατάστασης σε αυτή. Ο στόχος είναι να βελτιωθεί η λειτουργία και η συνολική χρησιμότητα των ICN αρχιτεκτονικών μέσα από τη χρήση της προσωρινής αποθήκευσης ως εγγενής αρχιτεκτονική λειτουργία. Τόσο ευκαιριακές μορφές προσωρινής αποθήκευσης όσο και υπηρεσίες διατήρησης αντιγράφων (CDN-like replication) εξετάζονται για την αντιμετώπιση αυτής της πρόκλησης. Έχουμε επινοήσει επίσης ένα σύνολο μηχανισμών προσωρινής αποθήκευσης, τους οποίους αξιολογούμαι μέσω κατάλληλων αναλυτικών μοντέλων, και οι οποίοι όχι μόνο βελτιώνουν σημαντικά τη συνολική ωφέλεια του δικτύου, αλλά και αποδεικνύουν την περαιτέρω δυναμική της συγκεκριμένης αρχιτεκτονικής που βασίζεται στο περιεχόμενο της πληροφορίας.

Αρχικά περιγράφουμε ένα πλαίσιο τριών φάσεων ως συμβολή στο πρόβλημα της αντιγραφής πληροφοριών στα ICN (replication) δίκτυα. Ο στόχος του προτεινόμενου πλαισίου είναι να ελαχιστοποιηθεί το συνολικό φορτίο της κυκλοφορίας στο δίκτυο μέσα από την εγκατάσταση ενός προκαθορισμένου αριθμού συσκευών αντιγραφής, και δεδομένου ότι κάθε συσκευή έχει πεπερασμένη δυνατότητα αποθήκευσης. Συγκεκριμένα, παρουσιάζουμε και αξιολογούμε ένα νέο αλγόριθμο για την επιλογή των θέσεων του δικτύου για την εγκατάσταση των συσκευών αντιγραφής. Επίσης προτείνουμε δύο εναλλακτικούς off-line μηχανισμούς για την εκχώρηση των αντίγραφων κάθε στοιχείου πληροφορίας μεταξύ των επιλεγμένων συσκευών αντιγραφής. Προτείνουμε επίσης μια κατανεμημένη αρχιτεκτονική διαχείρισης των αντιγράφων της πληροφορίας που δυναμικά αναθέτει/αλλάζει τα αποθηκευμένα αντίγραφα, προκειμένου να ελαχιστοποιηθεί το συνολικό κόστος της κυκλοφορίας του δικτύου που προκύπτει από τα αιτήματα των χρηστών. Συγκεκριμένα, προτείνουμε τέσσερις κατανεμημένους on-line αλγορίθμους διαχείρισης μνήμης, τους κατηγοριοποιούμαι ανάλογα με το επίπεδο συνεργασίας που απαιτούν μεταξύ των κόμβων του δικτύου και τους συγκρίνουμε με βάση την απόδοση τους, την πολυπλοκότητα τους καθώς και τον χρόνος σύγκλισης. Επίσης υπολογίζουμε το κάτω όριο του συνολικού κόστους της κυκλοφορίας του δικτύου για μια ορισμένη κατηγορία τοπολογιών δικτύου και δείχνουμε ότι οι προτεινόμενοι αλγόριθμοι διαχείρισης των αντιγράφων της πληροφορίας αποδίδουν πολύ κοντά σε αυτό το κάτω όριο.

Παρουσιάζουμε και αποσυνθέτουμε ένα πλαίσιο ευκαιριακής προσωρινής αποθήκευσης σε ένα σύνολο βασικών πολιτικών, παρουσιάζουμε σε κάθε υποσύνολο τις πιο γνωστές πολιτικές και προτείνουμε μια ICN πολιτική σε κάθε ένα από αυτά τα υποσύνολα. Παρουσιάζουμε επίσης ένα στοχαστικό μοντέλο που αποτυπώνει τη δυναμική των νέων προτεινόμενων πολιτικών και περιγράφουμε μια υλοποίηση του προτεινόμενου ευκαιριακού μηχανισμού αποθήκευσης τον οποίο αξιολογούμε πειραματικά στο πειραματικό δίκτυο PlanetLab. Επιπλέον προτείνουμε ένα νέο μηχανισμό δρομολόγησης που λαμβάνει υπόψη τις δυνατότητες της ευκαιριακής προσωρινής αποθήκευσης των κόμβων του δικτύου. Συγκεκριμένα παρουσιάζουμε έναν αλγόριθμο δυναμικού προγραμματισμού (DP) για τον υπολογισμό των διαδρομών ελάχιστου κόστους με βάση τα παρατηρούμενα μοτίβα που ακολουθούν οι επιθυμίες των χρηστών προκειμένου να ελαχιστοποιηθεί το συνολικό κόστος μεταφοράς της πληροφορίας στο δίκτυο που επιβάλλεται από τα αιτήματα τους.

Related Publications

The ideas presented in this thesis appear in the following publications:

Book chapter

[B.01] Vasilis Sourlas, Paris Flegkas, Dimitrios Katsaros and Leandros Tassiulas, “Content Replication and Delivery in Information-Centric Networks,” to appear in Advanced Content Delivery and Streaming in the Cloud by Wiley Publishers, USA.

Journal publications

[J.04] Vasilis Sourlas, Paris Flegkas and Leandros Tassiulas, “A Novel Cache Aware Routing Scheme for Information-Centric Networks,” submitted in Computer Networks Elsevier.

[J.03] Vasilis Sourlas, Lazaros Gkatzikis, Paris Flegkas and Leandros Tassiulas, “Distributed Cache Management in Information-Centric Networks,” to appear in IEEE Transaction on Network and Service Management (TNSM), 2013.

[J.02] Mohamed Diallo, Vasilis Sourlas, Paris Flegkas, Serge Fdida, and Leandros Tassiulas, “A Content-Based Publish/Subscribe framework for Large-scale Content Delivery,” in Computer Networks Elsevier, Volume 57, Issue 4, pp. 924-943, March 2013.

[J.01] Vasilis Sourlas, Paris Flegkas, Georgios S. Paschos, Dimitrios Katsaros, and Leandros Tassiulas, “Storage Planning and Replica Assignment in Content-Centric Publish/Subscribe Networks,” in S.I. on Internet-based Content Delivery, Computer Networks Elsevier, Volume 55, Issue 18, pp. 4021-4032, December 2011.

Conference publications

[C.13] Vasilis Sourlas, Paris Flegkas and Leandros Tassiulas, “Cache-Aware Routing in Information-Centric Networks,” in IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 582-588, Ghent, Belgium, 2013.

[C.12] Vasilis Sourlas and Leandros Tassiulas, “Effective Cache Management and Performance Limits in ICN,” in International Conference on Computing, Networking and Communications (ICNC 2013), pp. 955-960, San Diego, USA, 2013.

[C.11] Paris Flegkas, Vasilis Sourlas, George Parisis and Dirk Trossen, “Storage Replication in Information-Centric Networking,” in International Conference on Computing, Networking and Communications (ICNC 2013), pp. 850-855, San Diego, USA, 2013.

[C.10] Vasilis Sourlas, Paris Flegkas, Lazaros Gkatzikis and Leandros Tassiulas, “Autonomic Cache Management in Information-Centric Networks,” in 13th IEEE/IFIP Network Operations and Management Symposium (NOMS 2012), pp. 121-129, Hawaii, USA, April 2012.

[C.09] Dirk Trossen, Xenofon Vasilakos, Paris Flegkas, Vasilis Sourlas and George Parisis, "Mobility Work Re-Visited Not Considered Harmful," in IEEE WMCNT 2011, pp. 1-8, Budapest, Hungary, October 2011.

[C.08] Vasilis Sourlas, Lazaros Gkatzikis and Leandros Tassiulas, "On-Line Storage Management with Distributed Decision Making for Content-Centric Networks," in 7th Conference on Next Generation Internet (NGI) 2011, pp. 1-8, Kaiseslautern, Germany, June 2011.

[C.07] Mohamed Diallo, Serge Fdida, Vasilis Sourlas, Paris Flegkas and Leandros Tassiulas, "Leveraging caching for Internet-scale content-based publish/subscribe networks," in IEEE ICC 2011, pp. 1-5, Kyoto, Japan, June 2011.

[C.06] Vasilis Sourlas, Georgios S. Paschos, Petteri Mannersalo, Paris Flegkas and Leandros Tassiulas, "Modeling the dynamics of caching in content-based publish/subscribe systems," in 26th ACM Symposium On Applied Computing (SAC), Taiwan, March 2011.

[C.05] Vasilis Sourlas, Paris Flegkas, Georgios S. Paschos, Dimitrios Katsaros and Leandros Tassiulas, "Storing and Replication in Topic-Based Publish/Subscribe Networks," in IEEE Globecom 2010 Next-Generation Networking and Internet Symposium,Miami, USA, December 2010.

[C.04] Vasilis Sourlas, Georgios S. Paschos, Paris Flegkas and Leandros Tassiulas, "Mobility support through caching in content-based publish/subscribe networks," in 5th International Workshop on Content Delivery Networks (CDN 2010) in conjuction with 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2010), pp. 715-720, Melbourne, Australia, May 2010.

[C.03] Vasilis Sourlas, Georgios S. Paschos, Paris Flegkas and Leandros Tassiulas, "Caching in content-based publish/subscribe systems," in IEEE Globecom 2009 Next-Generation Networking and Internet Symposium, pp. 1-6, Hawaii, USA, December 2009.

[C.02] Vasilis Sourlas, Paris Flegkas, Georgios S. Paschos and Leandros Tassiulas, "Distribute, Store and Retrieve Management Policies on Wireless Ad-Hoc Networks using the Content Delivery Publish/Subscribe Paradigm," in proc. of 3rd IEEE Workshop on Autonomic Communications and Network Management - IM 2009 / ACNM 2009, pp 169-176, NY, USA, June 2009.

[C.01] Vasilis Sourlas, Paris Flegkas and Leandros Tassiulas, "Policy Distribution using the Publish-Subscribe Paradigm for Managing MANETs," in proc. of 11th IFIP/IEEE International Conference on Management of Multimedia and Mobile Networks and Services (MMNS 2008) held as part of Manweek 2008,pp 14-19, Samos, Greece, August 2008.

Contents

List of Figures

List of Tables

List of Abbreviations

AMP	......	Absorbing Markov Process
APID	......	Aggregated Publication Identifiers list
AS	......	Autonomous Systems
AT	......	Advertisement Table
BFS	......	Breadth-First Search
CBPS	......	Content-based Publish/Subscribe
CCN	......	Content-Centric Networking
CDN	......	Content Delivery Networks
CH	......	Cache
CM	......	Cache and Route Manager
CR	......	Content Router
CS	......	Content Store
CTMC	......	Continuous Time Markov Chain
DHT	......	Distributed Hash Table
DP	......	Dynamic Programming
DRG	......	Distributed Replication Group
DSR	......	Distributed Selfish Replication
DTN	......	Delay Tolerant Network
FIB	......	Forwarding Information Base
FIFO	......	First in First Out
ICN	......	Information-Centric Networking
IETF	......	Internet Engineering Task Force
ICP	......	Internet Cache Protocol
IP	......	Internet Protocol
ISP	......	Internet Service Provider
LFU	......	Least Frequently Used
LRU	......	Least Recently Used
MR	......	Mediation Router
NDN	......	Named Data Networking
NRS	......	Name Resolution Service
OSPF	......	Open Shortest Path First
P2P	......	Peer-to-Peer
PBR	......	Potential Based Routing
PIT	......	Pending Interest Table
PRT	......	Pending Request Table

QoS	......	Quality of Service
RENE	......	Rendezvous Network
RM	......	Resource Manager
RV	......	Rendezvous Node
SLA	......	Service-Level Agreement
SMVDHT	......	Scalable Multi-level Virtual Distributed Hash Table
ST	......	Subscription Table
TM	......	Topology Manager
URL	......	Uniform Resource Locator
VoD	......	Video on Demand
WRR	......	Weighted Round Robin

Chapter 1

Introduction

1.1 Motivation

Since mid 90's network operators are waiting a data tsunami, which fortunately did not come, but the tides of growth in the demand for data did help the Internet growth. However, today, the fear of data tsunami is slowly coming back [1]. The Cisco Visual Networking Index [2] reveals that data and media delivery have become immensely popular on the Internet. Particularly by 2016, global IP traffic will reach 1.3 zettabytes per year or 110.3 exabytes per month. Also, there will be nearly three networked devices per capita in 2016, up from over one networked device per capita in 2011. Driven in part by the increase in devices and the capabilities of those devices, IP traffic per capita will reach 15 gigabytes per capita in 2016, up from 4 gigabytes per capita in 2011. Additionally, global Internet video traffic will be 55 percent of all consumer Internet traffic in 2016, up from 51 percent in 2011. This does not include video exchanged through peer-to-peer (P2P) file sharing. Video exceeded half of global consumer Internet traffic by year 2011. The sum of all forms of video (TV, video on demand (VoD), Internet, and P2P) will be approximately 86 percent of global consumer traffic by 2016. This growth in demand is more dramatically depicted in Figure 1.1 which depicts the amount and the type of data that is transferred in the Internet in a minute [3].

The current Internet architecture focuses on communicating entities, leaving aside the exchanged information. However, trends, as shown in Figure 1.1, show that *what* is exchanged is becoming more important than *who* are exchanging it. As a result, the Internet is effectively moving from interconnecting machines to interconnecting information. Moreover, the location-centric model used by the underlying Internet routing paradigm, transferring of datagrams from one routable endpoint to another, limits the ability to fully utilize resources that are available along the route from the provider to the consumer(s), such as storage (e.g., for caching). Information-Centric Networking (ICN) has been proposed as a paradigm shift from the host-to-host Internet to a host-to-content one, or in other words from an end-to-end communication system to a native distribution network. This trend has attracted the attention of the research community, which has argued that content, instead of end-points, must be at the centre stage of attention.

DONA [4] was one of the first clean-slate information-centric proposals. DONA uses flat, self-

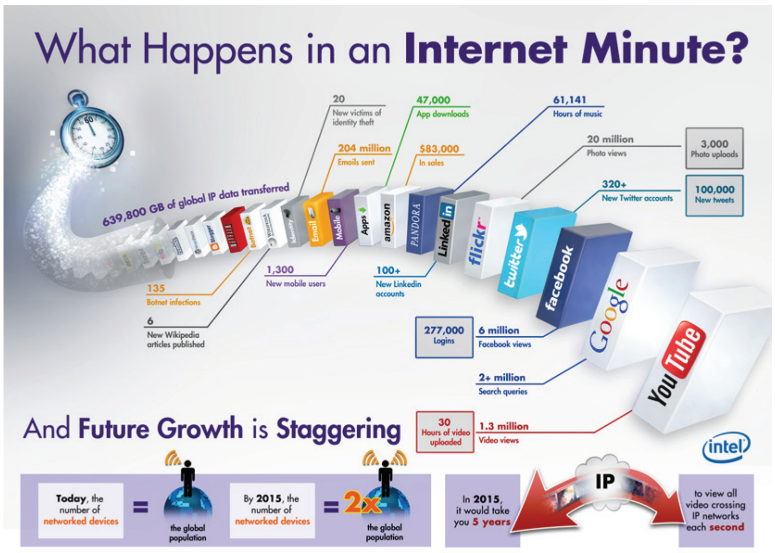
What Happens in an Internet Minute?
639,800 GB of global IP data transferred
20 New victims of identity theft
204 million Emails sent
47,000 App downloads
$83,000 In sales
61,141 Hours of music
20 million Photo views
3,000 Photo uploads
320+ New Twitter accounts
100,000 New tweets
135 Botnet infections
6 New Wikipedia articles published
1,300 New mobile users
100+ New Linkedin accounts
277,000 Logins
6 million Facebook views
2+ million Search queries
30 Hours of video uploaded
1.3 million Video views
And Future Growth is Staggering
Today, the number of networked devices = the global population
By 2015, the number of networked devices = 2x the global population
In 2015, it would take you 5 years
IP
to view all video crossing IP networks each second

Figure 1.1: Internet traffic in a minute.

identifying and unique names for information objects and binds the act of resolving requests for information to locating and retrieving information. The resolution layer is composed of interconnected *Resource Handlers* (RHs), forming a hierarchical name resolution and registration service upon the existing inter-domain relations. Registrations need to accumulate all the way up to tier-1 providers in order to guarantee that an existing name will be resolved. For that purpose, tier-1 providers must share state and have global knowledge. Content-Centric Networking (CCN) [5] has recently caught the attention of the research community. CCN proposes a name-based routing system for locating and delivering named data packets. The fundamental entities in CCN are Interest and Data packets. When a user wishes to receive data, he/she issues an Interest that contains the data name. The network propagates the Interests to the nearest data source (anycast) and then the requested item is delivered back to the user in the form of a Data packet. CCN uses names to identify content objects only; there is no notion of host name, point of attachment or path identifier. Content names follow a hierarchical form, similar to URLs or file-system paths and by definition, Interest and Data paths are symmetric. CCN applies an "one-to-one" Interest to Data packet principle in that an Interest can be satisfied by at most one Data packet. CONET [6] closely follows CCN, being differentiated from CCN in two ways: 1) it modifies the IPv4 packet header in order to make the architecture backwards compatible with existing routers and 2) it reduces the state maintained by routers via caching: a routing table has a finite size and a router stores the most recently used routing entries. When an arriving Interest specifies a name that is not present in the routing table, the router asks the domain's centralized naming system in order retrieve the routing entry.

In PURSUIT [7] and PSIRP [8], the design paradigm involves three separate elements and three separate functions: publishers, subscribers, and the REndezvous NEtwork (RENE) on one hand, and the functions of rendezvous, topology management/formation and forwarding on the other hand respectively. While the first three elements exist also in other candidate architectures under different names, the design principle of PURSUIT is to clearly distinguish the latter three functions. In more detail, publishers in PURSUIT advertise the availability of information by issuing a publication to the RENE. Similarly, subscribers are entities interested in consuming information who express their desire by issuing a subscription to the RENE for a specific piece of information. Therefore, the publish/subscribe paradigm is used here as an indirection to facilitate information resolution and discovery and not for actual data dissemination. Finally NetInf/SAIL [9] proposes a secure naming scheme composed of two main components: a naming scheme based on URLs and an information object structure holding information about a piece of content. These efforts promise, among other things, more flexibility in adapting to new services, efficiency improvements on lower layers, and native multicast support. However, crucial questions remain, like the management of ICN approaches which has not yet been addressed, focusing especially in the management of caching and route selection.

Caching performs replication of content to serve identical requests locally, and to prevent them from overutilizing network resources. A cache stores content on a storage device that is physically or logically closer to the user. In the area of information-centrism caching has been applied in two approaches: the CDN-like approach where dedicated storage devices are attached to the network and the in-network caching approach where all nodes of the network are potential caches. In the former approach every storage device (specific nodes) is responsible for storing information available from publishers enabling an asynchronous communication between publishers and subscribers in a transparent way. In the in-network caching approach each node opportunistically caches information traversing the node which can later transparently reply to subscriptions heading through it towards publishers or dedicated caches, potentially preventing saturation of up-stream network resources.

ICN inherently provides the opportunity of caching inside the network, due to its feature of naming content instead of end-hosts. Caching of chunks instead of whole files, raises issues of fair resource allocation and efficient content multiplexing, directly affecting packet-level network dynamics. In CCN content packets are cached by default in every router that the packet traverses [5]. In-network caching techniques are also being investigated in the IETF for potential standardization in the DECADE Working Group [10]. Little attention has been paid however, to caching based on collaboration incentives between different network elements.

Given this emergence of ICN oriented solutions, the relevant management needs in terms of performance have not been adequately addressed, yet they are absolutely essential for relevant network operation and crucial for those approaches to succeed. Performance management and traffic engineering approaches are also required to control routing. Routing functionality is completely missing from the current ICN design with only simple flooding or OSPF-like shortest path mechanisms having been proposed. This choice has been deliberately left open in order to allow routing solutions ranging from schemes potentially based on known protocols to innovative solutions best suited to the specific communication model.

In this thesis we have developed mechanisms that manage the routing processes by influencing the forwarding of interest/subscription packets and make caching decisions about where and which item to cache as well as influence the cache replacement policies. The aim is to improve the operations and overall utility of the ICN architectures through extensive and novel usage of caching as an inherent architectural function. Both CDN-like replication and in-network opportunistic caching is of interest in addressing this challenge. We have also devised a set of caching solutions, evaluated through proper analytical models, that not only dramatically improve the overall utility but also demonstrate the further potential of the architectural paradigm of information-centrism.

1.2 Synopsis

In this thesis we propose efficient caching algorithms deciding on the network location of caches, what information items to cache and influence the cache replacement policies. Particularly we investigate both CDN-like replication as well as in-network opportunistic caching. We also propose mechanisms and algorithms that manage the routing processes by influencing the forwarding of interest/subscription packets optimizing the network performance in terms of network load, congestion and delay.

In **Chapter 2** we initially present a brief, yet self-contained, introduction in Information-Centric Networking. We also describe a three phase framework as a contribution to the problem of information replication. The objective of the proposed framework is to minimize the total traffic load in the network subject to installing a predefined number of replication devices, and given that each device has storage limitations.

Next, in **Chapter 3** and **Chapter 4** we extensively describe and evaluate the three phase framework. Particularly, in Chapter 3 we present and evaluate a new algorithm for the selection of the location in the network to install replication devices. We also propose two alternative off-line mechanisms for the assignment of the replicas of each information item among the selected replications devices. In Chapter 4 we propose a distributed cache management architecture for ICN approaches that dynamically (re-)assigns information items to caches in order to minimize the overall network traffic cost imposed by the user requests. In more details, we derive four distributed on-line intra-domain cache management algorithms, categorize them according to the level of cooperation needed and compare them in terms of performance, complexity, message overhead and convergence time. We derive also a lower bound of the overall network traffic cost for a certain class of network topologies and show that the proposed cache management algorithms perform closely to the derived lower bound.

In **Chapter 5** we present and decompose an in-network opportunistic caching framework in a set of basic policies, present at each set the most known and propose an information-centric oriented policy at each one of them. We also propose a stochastic model that captures the dynamics of the newly proposed policies and describe a prototype implementation of the proposed opportunistic caching mechanism which is also evaluated through Planetlab experiments. Moreover, we present an enhancement of the opportunistic caching mechanism to enable subscribers mobility.

In **Chapter 6** we propose a new cache aware intra-domain routing scheme that dynamically computes the routes followed by each interest/subscription for each item and from each node in the network. Particularly, we present a dynamic programming (DP) approach for the computation of the minimum transportation cost paths based on the observed item request patterns in order to minimize the overall transportation cost imposed by the user requests. Finally, in **Chapter 7** we conclude our study and summarize the main findings of our work. Additionally, we discuss possible future directions. The three phase replication framework, the in-network opportunistic caching framework as well as the cache aware routing scheme are generic so that they can apply in almost every ICN approach.

Chapter 2

Replication Management Framework

In this chapter we initially present a brief, yet self-contained, introduction in Information-Centric Networking that can be used as reference for the rest of the thesis. Then we describe a three phase framework as a contribution to the problem of information replication in ICN.

2.1 Information-Centric Networking State of the Art

There exist different information-oriented networking models which are classified according to the semantic of the subscription language. *Channel-based* model allows information consumers to subscribe to publications originating from specific channels or feeds similarly to ATOM and RSS standards such as CORBA Event Service [11]. *Topic-based* model enables information consumers to register to a set of predefined topics such as TIBCO [12], while the *Information-Centric Networking* model supports subscriptions based on the actual content of the considered events.

ICN is a flexible communication model that meets the requirements of the content distribution trends in the Internet. ICN shifts the communication paradigm of the internetworking layer from end-points to information, where information is addressed by semantic attributes rather than origin and destination identities. ICN enables information consumers to register their information interests to a mediation entity that will prospectively retrieve content relevant to those interests. In the ICN there exist two different architectural designs. The first design contains implementations that are known as information-centric networks, e.g. PURSUIT [7], NDN/CCN [13] and SAIL [9], where information is explicitly named so that anybody who has relevant information can potentially participate in the fulfillment of requests for said information. The second architectural design, is known as *Content-Based Publish/Subscribe* (CBPS) and contains several research efforts that develop an overlay event notification service like IBMs Gryphon [14], Siena [15], Elvin [16], and REDS [17]. Their main distinguishing characteristic is that in the information-centric networks "*subscriptions*" act on the name of the object (that is, content is published and subscribed to by name), while in the CBPS design subscriptions can have broader request semantics (such as describing content with various tags and allowing subscriptions to relate to any content described with that tag). Also, the information-centric networks usually offers both a one-time "fetch" operation (retrieving content previously published under that name) and an ongoing "subscribe" operation (retrieving all future content published under that name). In contrast, most CBPS systems only support the latter [18] and in particular most of the CBPS implementations do not assume the presence of content servers (information items are not permanently stored and only active subscribers receive published items). CBPS supports subscriptions following an attribute/value schema or defined as vectors of keywords and provides efficient dissemination channels for a wide range of applications such as event notification, news dissemination or file sharing services. In this thesis we assume both architectural designs and we refer to them as Information-Centric Networking, but we clearly distinguish at each case which design is under consideration.

The core function of the ICN is the rendezvous function. The main purpose of the rendezvous function is to provide reachability from publishers and subscribers to data or service specific rendezvous points. The rendezvous function is required to scale to Internet-like network and data space sizes, and its operation needs to be efficient, measured in signalling overhead, overall latency and deployability (incremental deployment and stakeholder incentives). The rendezvous system defined in [8] is a composition of rendezvous networks that can be interconnected to form globally reachable rendezvous solution. The rendezvous networks are formed by rendezvous nodes (RNs) that are organized as a policy controlled inter-domain hierarchy. Large and geographically dispersed autonomous systems (ASes) may be covered by multiple rendezvous networks, but in the typical case a rendezvous network would be a collection rendezvous nodes from cooperating ASes.

ICN involves in general three types of entities: subscribers, receivers or information consumers, publishers or information providers and mediation routers (MRs). Each subscriber submits its interests by sending a subscription to the network where MRs acting as proxies are responsible for returning the corresponding matching pieces of data. In CCN an interest for an information item is routed towards the location in the network where that item has been published. At the nodes traversed on the way towards the source the caches of the nodes are checked for copies of the interested item. As soon as an instance of the item is found (a cached copy or the source item) it is returned to the subscriber along the path the interest came from. All the nodes along that path caches a copy of the item in case they get more requests for it.

In PURSUIT items are published into the network by the sources. Subscribers can then subscribe to items that have been published. The publications and subscriptions are then matched by the Rendezvous system. The matching procedure results in a Rendezvous Identifier (RI) that can be seen as an identifier for a communication channel. The RI in turn, can be resolved (within a scope) to a forwarding identifier that can be used for routing the information item through the forwarding network. In NetInf/SAIL information items are also published into the network. They are registered with a Name Resolution Service (NRS). When a subscriber wants to retrieve an item the request for the item is resolved by the NRS into a set of locators. These locators are then used to retrieve a copy of the item from the 'best' available source(s). Information providers can perform a wildcard registration of their principal in the NRS, so that queries can be directed to them without needing to register specific items.

2.2 A Three Phase Framework for the Replication Management

The main promise of current research efforts in the area of ICN is that of optimizing the dissemination of information within transient communication relationships of endpoints. Efficient replication of information is key to delivering on this promise. In this chapter, we look into achieving this promise from the angle of managed replication of information, where management decisions are made in order to efficiently place and assign replicas of information in dedicated storage devices attached to nodes of the network. Particularly, we present a framework for storage placement and replica assignment following both off-line as well as on-line approaches, whereas in the following two chapters we extensively describe and evaluate the three phase framework.

2.2.1 Introduction

In ICN information is explicitly labeled so that anybody who has relevant information can potentially participate in the fulfillment of requests for said information. Given the information-centric nature of the distribution utilizing information that is replicated across almost ubiquitously available storage devices is an almost natural thought. Optimized dissemination of information within transient communication relationships of endpoints is the main promise of such efforts and efficient replication of information, through the synergy of ICN with CDN techniques, is key to delivering on this promise.

While packet-level in-network opportunistic caching is one of the salient characteristics of ICN, proper cache placement and replica assignment still has an important role to play. CDN-like replication distributes a sites contents across multiple mirror servers. When a client is interested in a particular piece of information, his/her request is redirected to one of the existing replication points rather than requiring retrieval from the original publisher. Replication is used to increase availability and fault tolerance, while it has as side effect load-balancing and enhanced publisher subscriber proximity. Particularly, CDN providers strategically place surrogate servers and connect them to ISP network edges so that content can be closer to clients. Given the significant impact that content delivery has on the utilization of an ISP network, some work has recently started to investigate new models and frameworks to support the interaction between ISPs and CDNs.

In this chapter we present a three phase framework as a contribution to the problem of information replication in an ICN environment. The objective of the proposed framework is to minimize the total traffic load in the network subject to installing a predefined number of replication devices, and given that each device has storage limitations. The proposed framework is composed by three phases which manage the content and the location of each replication device in the network.

2.2.2 Related work

In the traditional context of CDNs, the placement problem is a thoroughly investigated problem. Particularly in [19] and [20] authors approached the placement problem with the assumption that the underlying network topologies are trees. This simple approach allows the authors to develop optimal algorithms, but they consider the problem of placing replicas for only one origin server. The placement problem is in fact an NP-hard problem [21] when striving for optimality, but there are a number of studies [22]-[27] where an approximate solution is pursued. Their work is also known as network location or partitioning and involves the optimal placement of k service facilities in a network of V nodes targeting the minimization of a given objective. In some cases, it can be shown that this problem reduces to the well known k-median problem.

The authors of [28] model replica assignment as a distributed selfish replication (DSR) game in the context of distributed replication groups (DRG). Under the DRG abstraction, nodes utilize their storage to replicate information items and make them available to local and remote users. The pairwise distance of the nodes is assumed to be equal, while our framework considers the generic case of arbitrary distances. In the context of DRG and under the same distance assumption a 2-approximation cache management algorithm is presented in [29]. Finally, in [30] authors develop a cache management algorithm aimed at maximizing the traffic volume served from the caches and minimizing the bandwidth cost. They focus on a cluster of distributed storages, either connected directly or via a parent node, and formulate the content placement problem as a linear program in order to benchmark the globally optimal performance.

More placement algorithms have been proposed in [21]. Particularly, authors formulate the problem as a combinatorial optimization problem and show that the best results are obtained with heuristics that have all the stores cooperating in making the replication decisions. Moreover, in [31] authors introduce a framework for evaluating placement algorithms. They classify and qualitatively compare placement algorithms using a generic set of primitives that capture several objectives and near optimal solutions. In most of the above approaches a similar cost function (optimize bandwidth and/or storage usage costs for a given demand pattern) is considered. Less attention has been given though to network constraints (limited storage capacity) and the possibility of reassigning items between caches as popularity and locality of users demand change.

In the area of replication in ICN in [32] a historic data retrieval publish/subscribe system is proposed, where databases are connected to various network nodes, each associated with a set of items to store. In [32] every information item is stored only once and no placement strategies have been examined. Finally, in the research area of investigating new models and frameworks to support the interaction between ISPs and CDNs, in [33] authors highlight that CDN providers and ISPs can indirectly influence each other, by performing server selection and traffic engineering operations respectively, and they investigate different models of co-operation between the two entities. In [34], authors propose a framework to support joint decisions between a CDN and an ISP with respect to the server selection process. This framework allows the ISP and the CDN to collaborate by exchanging some local information (network utilization from the ISP side and server conditions from the CDN side), so that it can result in better control of the resources. An ISP-supported CDN service has been proposed in [35] and [36], whereby content is stored and served from within ISP domains. This solution however can incur high operational costs, given that ISPs will have to maintain large storage capacities, and may thus be economically unviable.

2.2.3 Replication Framework

The proposed replication framework is composed by three phases namely the *Planning*, the *Off-line Assignment* and the O*n-line Replacement* phase which manage the content and the location of each replication device (the term *cache* is also used) in the network with objectives such as minimizing the content access latency from clients, maximizing the traffic volume served by the replicas and thus minimizing bandwidth cost and network congestion. In the Planning phase, the proposed framework selects those nodes of the network to place the replication devices while in the Off-line Assignment phase each information item is assigned, based on its popularity, at a subset of the selected replication devices so that the targeted objective is satisfied. Finally, the On-line Replacement phase dynamically reassigns information items in the replication devices based on the observed items changing request patterns. In order to support the proposed replication framework the three phases should be provided by relevant functional components. Regarding the Planning and the Assignment phase these components reside outside the network and run off-line algorithms at two different but long-medium time scales, while the Replacement phase is residing in components installed at each node of the network and run in real time scale.

The planning phase

The Planning phase takes as input the number of available replication devices an operator wishes to install, the network topology and a long term prediction of subscriptions in the network. It can run periodically deciding the optimal placement of the replication points at a long term time scale (e.g. once a year) or whenever the current location of them leads to an inefficient deployment due to significant subscriptions changes not successfully predicted. Performing and enforcing the decision of the planning component usually involves high-level business decisions since there is a high cost associated with moving a replication device to a different physical location or extending their number. In Chapter 3 an ICN oriented planning algorithm is presented for the selection of the replication points in the network based on the local demand for each item and the storage limitations of each replication device.

The off-line assignment phase

The component of the Off-line Assignment phase runs also periodically but at a medium/long term scale. It takes as input the outcome of the Planning phase regarding the locations of the replication devices installed in the network, the physical network topology and the medium/long term forecast. Replicas relocation can be enforced by instructing the replication devices to subscribe to a different set of information items. The instruction itself is realized as a publication of an information item to which replication devices are subscribed. In general, the replication points act both as publishers and subscribers for the information items they are instructed to store. They subscribe in order to receive new versions of the items, while they act as publishers for the same items to interested subscribers. That way, when a client subscribes to a specific piece of information one or more pub-

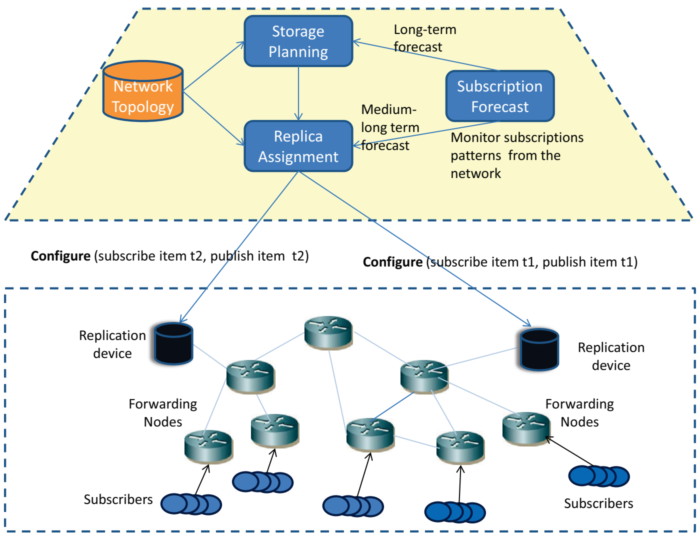

Figure 2.1: Architectural illustration of the Planning and the Off-line Assignment phases.

lishers/replication points are enabled, based on the operators policy, to publish the relevant data. The Assignment phase is also known as the Generalized assignment problem which even in its simplest form is reduced to the NP-complete multiple knapsack problem. In Chapter 3 we propose two alternative off-line mechanisms for the assignment of the replicas of each information item among the selected replications devices. Figure 2.1 illustrates the basic modules of the Planning and the Off-line Assignment phases.

The on-line replacement phase

As the provisioning periods of the Planning and the Off-line assignment phases can be quite long, the subscriptions pattern may significantly vary during that period. For that reason we introduce the On-line Replacement/Reassignment phase which enables the replacement of information items to the replication points to take place in real-time, based on the changing demand patterns of the users. Distributed components of that phase decide the items every replication point stores by forming a substrate that can be organized either in a hierarchical manner for scalability reasons or in a peer-to-peer organizational structure. Communication of information related to request rates, popularity/locality of information items and current replication points configuration, takes place between the distributed replacement components through an intelligent substrate.

Every replacement component, as depicted in Figure 2.2, should decide in a coordinated manner with other components whether to store an item. This may require the replacement of an already stored item, depending on the available space. The decision of this replacement of stored items is performed towards maximizing an overall network-wide utility function (e.g. the gain in network traffic), which means every node should calculate the gain the replacement of an item would incur. This approach assumes that every component has a holistic network-wide view of all the replication points

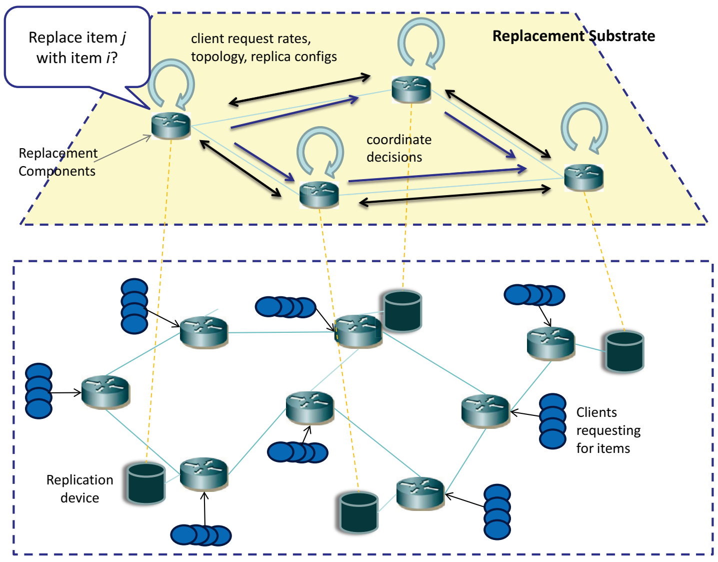
Replace item *j* with item *i*?
client request rates, topology, replica configs
Replacement Substrate
coordinate decisions
Replacement Components
Replication device
Clients requesting for items

Figure 2.2: Architectural illustration of the On-line Replacement phase.

configuration and relevant request patterns and this information should be exchanged periodically or in an event-based manner when a component changes the configuration of its attached replication device.

Other approaches can also be realized in which the components base their decisions on a local view of the users demand for specific items but coordinate to maximize the overall network gain, as well as solutions where components act selfishly aiming at maximizing their own local utility. Since all the above decisions are made in a distributed manner, uncoordinated decisions could lead to sub-optimal and inconsistent configurations. Coordinated decision making of a distributed solution can be achieved through the substrate mechanisms, by ensuring that components change the configuration of each replication device in an iterative manner i.e. one at a time and not autonomously in a potentially conflicting manner.

Any distributed on-line mechanism that applies in this third phase should capture the volatile environment under consideration. All of them should be adaptive to popularity and locality changes by fetching new items at a replication device and replacing existing ones. We envision three classes of algorithms that could be applied in the On-line Replacement phase that differ in the amount of information that needs to be communicated through the substrate, the required level of coordination among the components, and the performance objective. We briefly present them in order of decreasing complexity, in terms of the induced computational/communication overhead, while in Chapter 4 we extensively present and evaluate specific algorithms for each one of the classes.

The first class, henceforth called *cooperative*, aims at minimizing the overall network traffic. This requires that every component needs a holistic network-wide view of the request patterns and the current replication points configuration. In addition, since each replacement decision affects the whole network, some cooperation in the decision making is required. A cooperative algorithm in the context of distributed replication groups (DRG) is presented in [29]. Under the DRG abstraction, where the pairwise distance of the nodes is assumed to be equal, nodes utilize their caches to replicate information items and make them available to local and remote users. The algorithms of the cooperative class requires at each iteration each component of the network to compute the relative gain of every possible replacement and through appropriate message exchange to cooperate for the final replacements. In other words every component participates at each iteration in the execution of the algorithm but only one every time (the one with the maximum relative gain) performs valid replacements.

The second class, henceforth called *holistic* also aims at minimizing the overall network traffic and hence requires the same amount of information. On the other hand, in the holistic class there is no need for coordination of the actions of the components and the required decisions are made in an autonomous manner by each one individually. The holistic class is of similar nature as the cooperative and towards the same objective. Its distinguishing characteristic though is that each component operates in its own performing replacements on the respective replication point. Particularly, only one component at each iteration performs valid and beneficial replacements towards a specific objective. In [28] a holistic algorithm in the context of DRG is also presented. Additionally in [30] authors develop a replication management algorithm for a cluster of distributed replicas (CDNs) aimed at maximizing the traffic volume served from the replication devices and minimizing the bandwidth cost.

In both the cooperative and the holistic class it can be shown that since any change performed in the replication points configuration decreases the overall network traffic, the proposed algorithms finally converges to an equilibrium point where no further improvement is possible. The algorithms do not necessarily converge to the optimal assignment, but to a local minimum of the objective given the initial configuration. In both the holistic and the cooperative classes every node needs to acquire global knowledge regarding the demand pattern for the decision making. However, in highly dynamic environments the amount of information that needs to be circulated among the components becomes significant, causing thus non-negligible communication overhead. Even worse the required information may not be available on time, making hence such an approach inapplicable. For such scenarios we present an alternative class of algorithms called *myopic*, where each component needs global knowledge of the items stored in the network, but only local knowledge regarding demand. Also, in the myopic class each component acts autonomously and has the objective of minimizing the traffic related to its own demand. In the proposed algorithmic classes the replacements are based on the real time observed items request patterns such as their popularity and locality and not in static off-line predictions. It is evident that the network wide knowledge and cooperation give significant performance benefits and reduce significantly the time to convergence, but at the cost of additional message exchanges and computational effort.

Chapter 3

Storage Planning and Off-Line Replica Assignment

In various implementations of the CBPS architectural design (e.g. REDS, Siena), information items are not permanently stored and only active subscribers receive published items. However, in a dynamic scenario, where users join the network at various instances, a user may be interested in content published before its subscription time. In this chapter, we introduce a mechanism that enables storing in such systems. Furthermore, we propose a new storage placement and replica assignment algorithm which differentiates the information items based on their popularity and minimizes the subscribers response latency and the overall traffic of the network. We also present and compare two off-line replica assignment alternatives and examine their performance when both the locality and the popularity of users request change. The performance of our proposed placement and replica assignment algorithm and the proposed storing mechanism is evaluated via simulations. The proposed mechanism is compared with mechanisms from the CDN context and performs as close as 1%-15% (depending on the conducted experiment) to a greedy (near optimal) approach installing up to 3 times less replication devices in the network and providing the necessary differentiation among the classes of the content.

3.1 Introduction

In the CBPS architectural design, any information item is guaranteed to reach all interested subscribers as long as their subscriptions are known to the network at publish time, assuming stable topology and no queuing overflows. However, in a dynamic distributed environment, subscribers join and leave the network over time, and it is possible that a subscriber joins the network after the publication of an interesting items, so it is not possible for a new subscriber to retrieve already published items that match his/her subscription. Therefore, enabling the retrieval of past published information items by means of replication is one of the most challenging problems.

Content delivery servers ("surrogate servers" in CDNs or simply "replicas" in this work) replicate the whole content of a given server and target to speeding up the delivery of content by reducing the load on the origin servers and the network itself. When a client is interested in a piece of information of a given server, his/her request is redirected to one of the existing replicas (e.g., the closest one or the one satisfying other criteria such as the load of the candidate replication point). Since replicas serve only a portion of the total requests and are placed closer to the client, clients are served faster. A client's request is redirected to a replica only if that replica replicates the targeted server, otherwise the request is directed and served by the server itself.

In this chapter we:

- Enhance the CBPS communication paradigm with an advertisement and a request/response mechanism so that replicas can advertise what they have stored and subscribers can retrieve it.
- Propose a new algorithm for the selection of R replication points among the V nodes of the network ($R < V$) based on: a) the locality and the popularity of the interests for each information item, b) the targeted replication degree of each item (as replication degree we name the number of replicas k_m ($1 \leq k_m \leq R$) of item $m \in M$ among the stores, which is analogous to its popularity) and c) the storage capacity (limitation) L of each replication device.
- Propose two alternative mechanisms for the assignment of the replicas of each item among the selected replication points.
- Evaluate through simulations our design of the storing mechanism and the new placement and replica assignment algorithm.

The objective of our scheme is to minimize the total traffic load in the network subject to installing the minimum number of replication devices and given that each replica has storage limitations. Of course, the proposed storing scheme is generic and can be applied in every information-centric approach, but in this chapter the whole analysis and the proposed mechanisms were designed assuming the CBPS architectural design.

The rest of the chapter is organized as follows. In Section 3.2 we describe the problem under investigation, while in Section 3.3, we shortly describe the CBPS architecture and present the proposed advertisement and request/response mechanism. The new algorithm for the selection of the replicas' location and the replica assignment of the items is presented in Section 3.4. Section 3.5 is devoted to performance evaluation via simulations. Finally, we conclude the chapter in Section 3.6.

3.2 Problem Formulation

In this chapter, we assume an network with arbitrary topology of V nodes. M different information items should be stored at R replication points, where each replication point has the capability to store L different items. Each item $m \in M$ should be replicated k_m times. Requests for the items are generated at various nodes and they trigger the transfer of the requested item from a store to the node where the request was generated. The proposed mechanism is composed by two phases, typical in any network management task, namely the *Planning* and the *Off-line Assignment* phase. In the *Planning* phase, the proposed mechanism selects R points out of the V nodes of the network to place the replicas, while in the *Assignment* phase, each item $m \in M$ is assigned at exactly k_m different replicas with the target to minimize the total traffic load in the network.

Generally, in a real implementation the *Planning* of stores changes rarely, since it requires the reallocation of the stores among the network nodes. On the other hand, the *Assignment* of the items is more flexible and the storage provider (e.g. a CDN provider) is able to reassign the items among the replication points when the locality and the popularity patterns change is such a way that the performance of the network is degraded with the existing configuration. Of course, a reassignment requires the calculation of the new places for each item and the transfer of items to those locations, but as shown later in the performance analysis, it is an efficient way to maintain the performance of the system at high levels without re-planning the whole network.

The storage capacity of each replication point usually refers to TBytes but for simplicity we assume here that items are of the same size. The L parameter is a limitation introduced by the storage providers and refers to the maximum storing capability of each store in the network. On the other hand, the k_m parameter (replication degree of each item) is a limitation introduced by the content provider and refers to the number of replicas that the content provider is willing to pay for. Finally, the R parameter refers to the number of stores that a storage provider should install in the network to serve the storage demands of the content provider.

3.3 Enabling Replication

In this chapter, we consider a network which uses the subscription forwarding routing strategy [15], where the routing paths for the published items are set by the subscriptions, which are propagated throughout the network so as to form a tree that connects the subscribers to all the nodes in the network. In that scheme, publishers join the network when they have something to publish, therefore in our approach the entity of the origin server does not exist.

When a subscriber issues a subscription, a packet containing the subscription filter is sent to the node the subscriber is attached to. The filter is inserted in a Subscription Table (`ST`), together with the identifier of the subscriber. Then, the subscription is propagated by the node, which now behaves as a subscriber with respect to the rest of the dispatching network, to all of its neighboring nodes. In turn, the neighbors record the subscription and re-propagate it towards all further neighboring nodes, except for the one that sent it. Finally, each node in the network has a `ST`, in which for every neighboring node there is an associated set of filters containing the subscriptions issued by them.

3.3.1 Advertisement and Request/Response Mechanism

In this section, we present the advertisement and the request/response mechanism, which provides a CBPS implementation with the ability to store and retrieve information published in the past and make it available for future clients. Particularly, we will present the new mechanisms through the example of Figures 3.1 and 3.2.

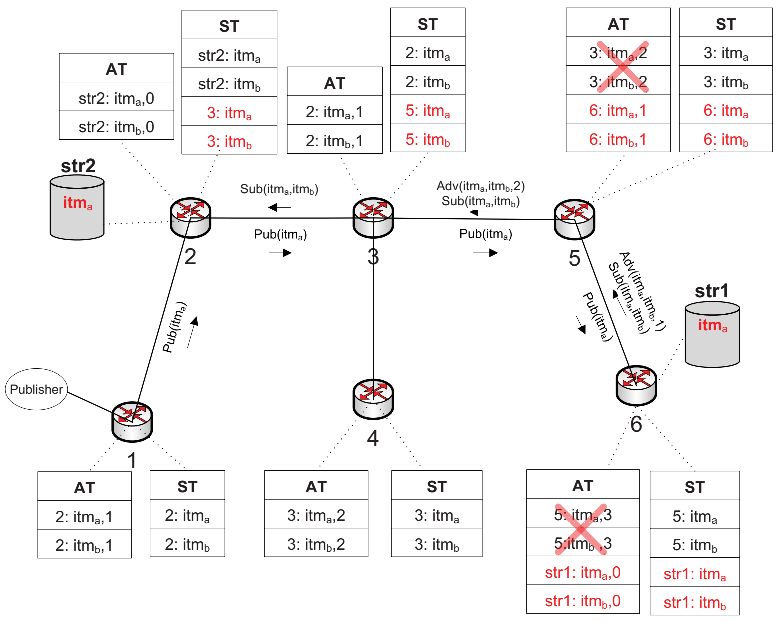

Figure 3.1: Advertising and Storing of information.

In order to retrieve stored information, we enhance the CBPS architectural model with three additional types of packets (besides the already existing `Publish()` and `Subscribe()`), `Advertise()`, `Request()` and `Response()`. We also add to the system a new feature called `Advertisement Table` (`AT`), similar to `ST`, which is used to store advertisements. When a new replication point "*str1*" is installed at node 6 (Figure 3.1), it issues a set of `Subscribe()` packets with the items that is willing to store (itm_a and itm_b in the given example). In that way, it acts as a subscriber to future publications matching the subscribed items and, each time a relevant publication occurs (i.e. publisher attached to node 1 publishes item itm_a), it stores the item (the item is also stored to "*str2*"). The "*str1*" also issues a set of `Advertise()` packets, which contains the items that stores and the distance in hops from the store (the distance attribute is built hop by hop). Advertisements are treated similarly to subscriptions and form a tree that connects the "*str1*" to all the nodes in the network. Advertisements are inserted in the (`AT`). Coverage also occurs with advertisements, as with subscriptions, but in a slightly different way. Particularly, when a node receives an advertisement, checks in the distance field and if the distance is equal to another entry (for the same item), it adds the advertisement to the `AT` and stops forwarding the advertisement. Keeping more than one entry for the same item in an `AT`, enables load balancing capabilities to requests passing from that particular node. On the other hand, when a node receives an advertisement for a replication node which is closer compared to the other replicas already in the `AT`, it adds the advertisement to the `AT`, removes the previous entries and forwards it further (nodes 5 and 6 in Figure 3.1). Finally, when a node receives an advertisement for a replica which is further compared to the other replicas already in the `AT`, it simply stops the forwarding of the advertisement without changing the entries of the `AT` (node 3 in Figure 3.1).

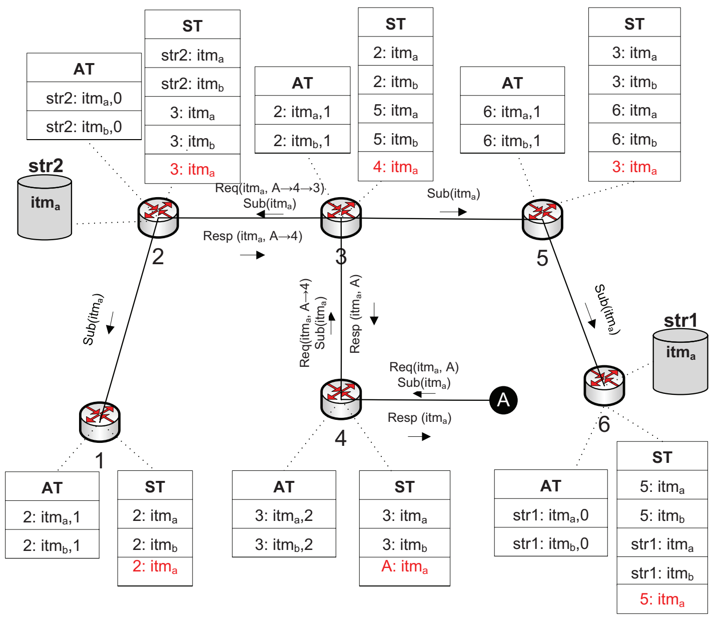

Figure 3.2: Retrieval of stored information using the request/response mechanism.

When a subscriber (subscriber A in Figure 3.2), is interested in retrieving stored content apart from subscribing (if he/she is also interested for future publications) he/she issues a request by sending a `Request()` packet. Source routing is used for the forwarding of the `Request()` (the path is being built hop by hop and is included in the `Request()` header). Node 4 upon receiving the `Request()` checks in its `AT` for entries matching the requested item (itm_a in this case). The node forwards the `Request()` to the neighbor who had advertised the matching item and is closer to the subscriber (in this example node 3 and finally node 2). Finally, "str2" receives the `Request()`, and initiates a `Response()` including the requested item.

A `Response()` packet carries the information item as well as the sequence of nodes carried by the initiating `Request()` (source routing). When a node receives a `Response()` message it pops off its identifier from that sequence and forwards it to the first node of the remaining sequence. In the end, subscriber A will receive the requested item.

3.4 Placement and Replica Assignment Strategy

We use as the base of our placement and replica assignment scheme, algorithms presented in the context of CDN networks. Particularly in [21] and [22], authors developed several placement algorithms that use workload information, such as latency (distance from the storage points) and request rates, to make the placement decision. Their main conclusion is that the so called "*greedy*" algorithm that places replicas based upon both a distance metric and request load, performs the best and is very close to the optimal solution.

3.4.1 Greedy algorithm

Here, we briefly present the greedy algorithm assuming that there exists only one item in the network. We let r_i be the demand (in reqs/sec) from subscribers attached to node i. We also let p_{ij} be the percentage of the overall request demand accessing the target server j (traditional placement algorithms replicate a specific origin server) that passes through node i. Also we denote the propagation delay (hops) from node i to the target server j as d_{ij}. If a store is placed at node i we define the Gain to be $g_{ij} = p_{ij} \cdot d_{ij}$. This means that p_{ij} percentage of the traffic would not need to traverse the distance from node i to server j decreasing the overall network traffic by:

$$d_{ij} \cdot \sum_{l=1}^{N} \overline{r}_l$$

where

$$\overline{r}_l = \begin{cases} r_l & \text{if } i \text{ is on the path from } l \text{ to } j \\ 0 & \text{otherwise.} \end{cases}$$

The greedy algorithm chooses one replication point at a time (we need k replicas out of the V nodes of the network). In the first round, it evaluates each of the V nodes to determine its suitability to become a replica of server j. It computes the Gain associated with each node and selects the one that maximizes the Gain. In the second round, searches for a second replication point which, in conjunction with the replica already picked, yields the highest Gain. The greedy algorithm iterates until k replicas have been chosen to replicate server j.

3.4.2 Modified Greedy Algorithm

In the architectural design that we assume here, the notion of an origin server - which is vital for the greedy algorithm - does not exist. Publishers join the network, publish their content and disappear. So in order to obtain the location of the replicas we modify the greedy algorithm. Particularly, we repeat the above procedure V times assuming each time that the targeted server j is a different node of the network. We get in that way V vectors of k possible stores. Precisely, each vector has V elements with k ones in the index of the selected stores and $V-k$ zeros in every other place. For example, vector $[0\ 0\ 0\ 1\ 0\ 1]$ means that of the 6 nodes of the network the selected $k=2$ possible stores are nodes 4 and 6. Finally, we select as our replication points those k nodes that appeared more times in the per element summation of the V vectors, and install at each one a replica following the mechanism described in Section 3.3.1. The modified greedy algorithm presented here assumes uniform distribution of the probability among the V nodes of the network that publications could occur. Of course, other forms of probability distributions could be used, and each vector should be first weighted with its probability before the per element summation of the V vectors.

V	: number of nodes in the network
R	: $(R < V)$ number of replication points in the network
M	: number of information items in the network
L	: storage capacity of each replication point in the network
r_i^m	: request rate for item $m \in M$ in node $i \in V$
k_m	: $(k_m \leq R)$ replication degree of item $m \in M$
w_m	: weight of item m
w'_m	: relative weight of item m
$\mathcal{S}$	: vector of replication points
$\mathbf{s}_m$	: possible replication points vector for item m

Table 3.1: Parameters used by the placement algorithm and its assignment alternatives

3.4.3 Placement and Replica Assignment Algorithm

Here, we use the modified greedy algorithm described above for the case where in our network exist M different items. Next, we present the Steps of the proposed algorithm side by side with the example of Figure 3.3 (Table 3.1 contains all the useful parameters required by the proposed algorithm):

Step 1: For each item $m \in M$ we execute the modified greedy algorithm presented in Section 3.4.2 and we get M vectors of possible replication points $\mathbf{s}_m$. Regarding the example we get: $\mathbf{s}_a = [0\ 3\ 5\ 0\ 2\ 2]$, $\mathbf{s}_b = [0\ 2\ 5\ 0\ 5\ 0]$, $\mathbf{s}_c = [0\ 2\ 5\ 0\ 5\ 0]$ for the three items accordingly. The $[0\ 3\ 5\ 0\ 2\ 2]$ means that out of the $V = 6$ executions of the modified greedy algorithm, node 2 appeared 3 times, node 3 appeared 5 times and so on.

Step 2: Each vector $(\mathbf{s}_m)$ is weighted by $w_m = \frac{\sum_{i=1}^{V} r_i^m}{\sum_{i=1}^{V}\sum_{m=1}^{M} r_i^m}$. w_m shows the significance (popularity) regarding the traffic demand of each item in the network. The weights for the given example are: $w_a = 17/50 = 0.34$, $w_b = 27/50 = 0.54$, $w_c = 6/50 = 0.12$.
We obtain the following weighted vectors:
$w_a \cdot \mathbf{s}_a = [0\ 1.02\ 1.7\ 0\ 0.68\ 0.68]$, $w_b \cdot \mathbf{s}_b = [0\ 1.08\ 2.7\ 0\ 2.7\ 0]$, $w_c \cdot \mathbf{s}_c = [0\ 0.24\ 0.6\ 0\ 0.6\ 0]$.

Step 3: We select as our replication points those R nodes that appeared more times in the per element weighted summation of the M vectors. We call that vector the *vector of replication points* $\mathcal{S}$. The per element summation of the above three vectors into a single vector gives $[0\ 2.34\ 5\ 0\ 3.98\ 0.68]$ meaning that the final $R = 3$ replication points in $\mathcal{S}$ are nodes 3, 5 and 2.

Step 4: For each item m, starting from the most significant (based on the weight), we assign k_m replicas following the procedure below:

- For each entry in the $\mathbf{s}_m$ of item m calculated in Step 1 assign a replica if that entry also appears in the $\mathcal{S}$, calculated in Step 3, and only if in that replication point has been assigned less than L (storage capacity) items until we get k_m replicas (replication degree of item m).

In the example starting from item b then item a and finally item c (based on their weights) we assign them to $k = 2$ replication points. Item b is assigned to nodes 3 and 5 which were the

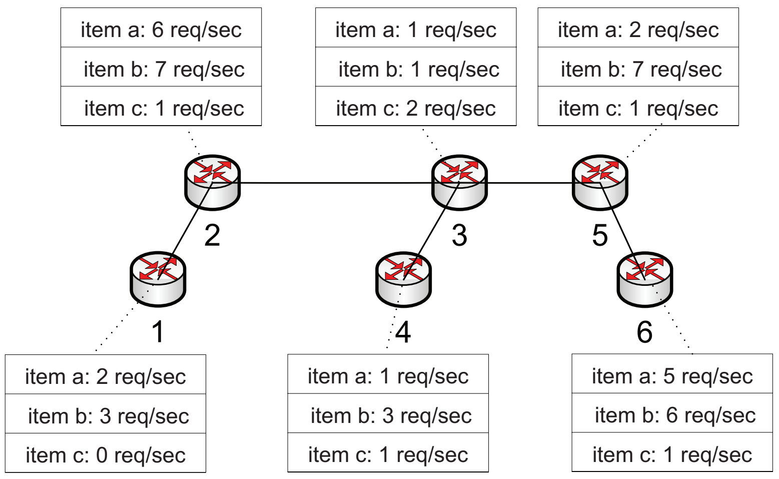

Figure 3.3: Topology and workload information per each item together with, $k_m = k = 2$, $L = 2$ and $R = 3$ form the inputs of the placement algorithm.

nodes for item b appeared more times in Step 1. Item a is also assigned to nodes that were produced by Step 1, nodes 2 and 3, while item c is assigned to nodes 2 and 5. Node 5 was among the most popular selections produced by Step 1 while node 2 was the only store in $\mathcal{S}$ with less than $L = 2$ assignments.

Step 4 of our algorithm is also known as the *Generalized Assignment Problem* which even in its simplest form is reduced to the NP-complete multiple knapsack problem. In this chapter, for the solution of the assignment problem we use the heuristic approaches described above and in Section 3.4.5, while more approaches could be found in literature [37].

3.4.4 Cost Model

Steps 1-3 of the proposed algorithm described above comprise the *Planning* phase of the algorithm while Step 4 is the *Assignment* phase. In this section, we present the cost model of the *Assignment* phase, which as mentioned above is an NP-complete problem. The access of an information item stored in replication point x by a subscriber attached at node y generates a traffic load equal to the length (number of hops) of the path from x to y. Given that we wish to optimize the total traffic load, the access scheme is that we always access the closest store (shortest path) among those holding the specific item. Thus given the access mechanism, we seek to decide the replica assignment of each item.

Let T be the traffic load corresponding to any storage configuration. For that storage configuration we can write:

$$T = \sum_{m=1}^{M} T_m \tag{3.1}$$

where T_m is the traffic load corresponding to configuration of item m only.

We then have,

$$T_m = \sum_{n=1}^{k_m} \sum_{l \in \mathcal{N}_n^m} r_l^m \cdot d_{ln} \tag{3.2}$$

where $\mathcal{N}_n^m$ is the collection of nodes accessing item m from its replication point at node n, r_l^m is the request rate for item m from node l and d_{ln} is the distance (in hops) from node l to node n.

And for the overall network traffic from Equations 3.1 and 3.2, we get:

$$T = \sum_{m=1}^{M} T_m = \sum_{m=1}^{M} \sum_{n=1}^{k_m} \sum_{l \in \mathcal{N}_n^m} r_l^m \cdot d_{ln} \tag{3.3}$$

The minimization of the overall traffic cost is given by the minimization of the following constrained nonlinear multivariable function.

$$\min\left(T\left(k_1, k_2, \ldots, k_m\right)\right) \text{ such that } \begin{cases} \sum_{m=1}^{M} k_m \leq L \cdot R \\ 1 \leq k_m \leq R, \forall m \in M \end{cases} \tag{3.4}$$

3.4.5 Alternatives on the Assignment Phase

In this section we describe an alternative assignment mechanism which is similar to the Weighted Round Robin (WRR) scheduling discipline. In Section 3.4.3 items were assigned sequentially based on their weights. Particularly, the item with the largest weight was assigned first, then the item with the second largest weight and so on. In the WRR alternative the sequence of the assignment does not change but the number of storing points that each item is assigned to is based on its relative weight. The relative weight w'_m of item m is $w'_m = \left\lceil \frac{\sum_{i=1}^{V} r_i^m}{\min_{\{m \in M\}} \left\{ \sum_{i=1}^{V} r_i^m \right\}} \right\rceil$. This means that the w' of the less weighted item is equal to one. The w' of all items generate an integer vector of the form $[w'_1, w'_2, ..., w'_M]$ where w'_1 is the relative weight of item 1 and so on (e.g. [3 1 2 2] means that out of the four items, item 2 is the the one with the smallest weight while items 3 and 4 are twice as large as item 2 and item 1 is the largest and its weight is three times larger than item 2). The WRR alternative of the assignment procedure assigns at each round ξ

$$k_m^{\xi} = \min\left\{ w'_m \cdot k_m,\ k_m - \sum_{\xi'=0}^{\xi-1} k_m^{\xi'} \right\}$$

where

$$k_m^0 = 0, \quad \forall m \in M$$

replicas of each item until all items are assigned to k_m different replication points.

In the example of Figure 3.3, the vector of the relative weight of the items is [3 5 1]. Of course the assignment procedure of WRR alternative in that example is the same to the assignment procedure described in Section 3.4.3 since $k_m = k = 2$ but in the case that $k = 6$ then the assignment of WRR would have been:

Round 1: [3 5 1] replicas for each item.
Round 2: [3 1 1] (item 2 has already been assigned to 5 replication points).
Round 3: [0 0 4] (items 1 and 2 have been assigned to $k = 6$ replication points).
The WRR alternative is fairer to the less weighted items and as shown in the performance evaluation this lead to better performance regarding the clients' perceived delay and the overall network traffic.

3.5 Performance Evaluation

In this section, we evaluate the proposed replication mechanism using a discrete event simulator. The simulator is written in MATLAB based on the event-driven technique for continuous-time modeling. Before implementing the modified greedy algorithm and the two alternative assignment algorithms we made sure that the implemented greedy algorithm is inline with the algorithms presented in [21] and [22] (the performance of our greedy implementation is inline with the plots of those two greedy implementations). V nodes are organized in a tree topology and subscribers dynamically request on each node i for stored items with rate r_i^m different for each item m. We assume that in our network exist M items and based on the set of experiments each item should be either replicated at least min_{k_m} times or a predefined number of R replication points should be placed and appropriately assigned to the items. Also, each replication point has a capacity of L different items. Finally, the assignment of replicas to the items is based on their actual weight w_m with the constraint that at least min_{k_m} replicas should be assigned to each item m.

It is widely acknowledged that information-centric research lacks public data sets for meaningful evaluation. Thus, synthetic workload generation is widely accepted in the field, under the assumption that the workload generated meets a set of realistic assumptions. Each item is characterized by two parameters: *popularity* and *locality*. Popularity refers to the request rate related to an item and locality to the region of the topology likely to originate requests. p_m (respectively l_m) denotes the popularity (respectively the locality) associated to an item m. Popularity and locality values are computed using a Zipf law of different exponents s_p and s_l respectively. Requests are issued from a set of nodes computed using l_m. Particularly $\lceil l_m \cdot V \rceil$ nodes are potential issuers of requests related to item m. This set of nodes is computed by choosing a random central node and $\lceil l_m \cdot V \rceil - 1$ additional nodes among the closest nodes to the central node (executing a Breadth First Search algorithm).

Having selected the R replication points and assigned to them the M information items using our two assignment alternatives ("*p/s_seq*" for the sequential assignment mechanism and "*p/s_wrr*" for the weighted round robin-like assignment mechanism) we let the system operate under the dynamic client environment. We compare it firstly to the case where each item is assigned to the k_m replication points produced by the first step of the placement algorithm ("*grd_opt*") described in Section 3.4.3 disregarding of the storage capacity and the total number R of used replicas, and secondly to the case where there is no differentiation among items during the selection of the R replication points and the final assignment of the items to k_m replicas is random ("*rnd*"). The metrics we are interested in are:

- The *Network Traffic* (in $resps \cdot hops/sec$) after the completion of the placement/replication

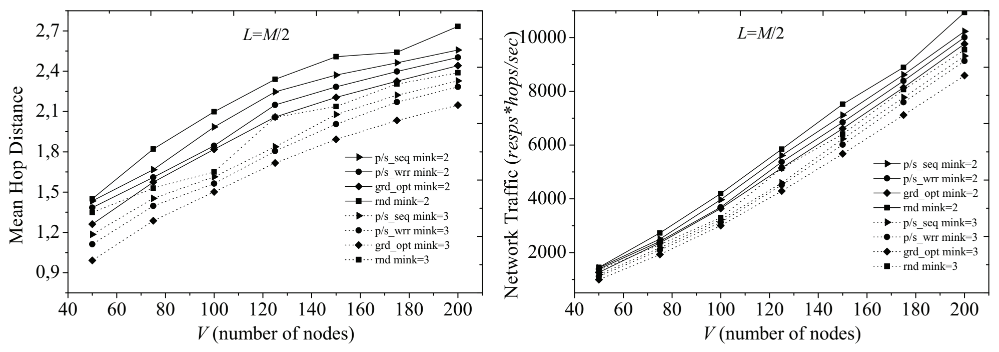

Figure 3.4: Performance of the proposed placement algorithm (both assignment alternatives "seq" and "wrr") compared to the "grd_opt" and the "rnd" vs. the number of the nodes in the network.

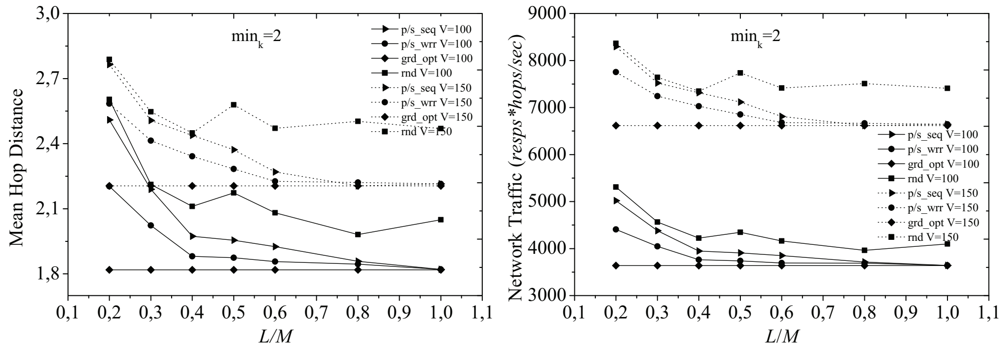

Figure 3.5: Performance of the proposed placement algorithm (both assignment alternatives "seq" and "wrr") compared to the "grd_opt" and the "rnd" vs. the storage capacity of the replication points in the network.

algorithm.

- The *Mean Hop Distance* which corresponds to the mean number of hops between a responding replication point and the subscriber issuing the request. This metric is indicative of the response latency as a function of hops in the network.

The above metrics are random variables and we estimate their mean by simulating thousands of observations. We run two sets of experiments; one evaluating both the planning and the assignment phases of the proposed framework and one evaluating only the reassignment of items after an initial planning.

3.5.1 Overall Evaluation of the Placement and the Replica Assignment Algorithm

In the first set of experiments we conducted two subsets of experiments one assuming a predefined minimum replication degree for each item and one assuming a predefined number of replication devices that should be installed in the network.

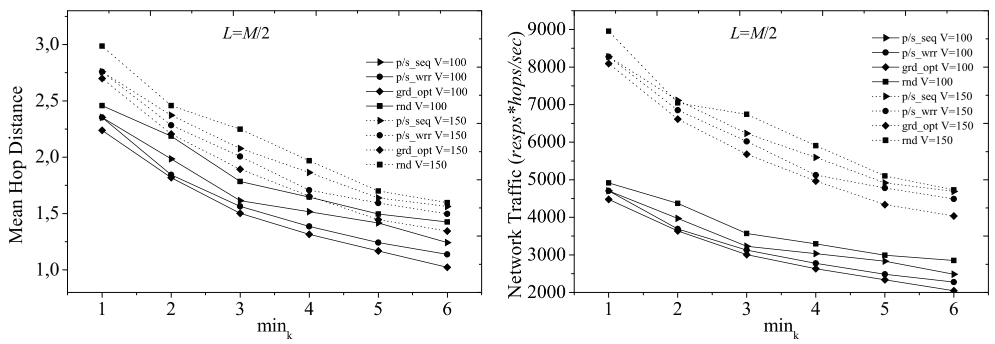

Figure 3.6: Performance of the proposed placement algorithm (both assignment alternatives “seq” and “wrr”) compared to the “grd_opt” and the “rnd” vs. the minimum replication degree of the items in the network.

Predefined minimum replication degree

In this subset of experiments, we assumed that the Zipf’s exponent value of the popularity is $s_p = 1$ while we assumed uniform locality among the items. Uniform locality implies that requests are generated from every node in the network for every item or else, the neighborhood of interest for each item is the whole network. We also assumed that $min_{k_m} = min_k = 2$ (minimum replication degree). We mentioned above that the assignment is weighted based on the w_m of each item, meaning that the number of replicas of each item is given by $k_m = \left\lceil \frac{w_m}{w_{m'}} \cdot min_k \right\rceil$ where $m' \in M$ is the less weighted item. Also in our network exist $M = 1000$ different items and the clients’ request rate per item m is $25 \cdot p_m$ requests/second from each node of the network. Particularly, we run three different experiments, one varying the number of nodes in the network, one varying the storage capacity of each potential replication point and one varying the minimum replication degree min_k of the items in the network.

Figures 3.4 - 3.6 show the mean hop distance and the network traffic for each one of the three different experiments. The proposed algorithm behaves better than the “rnd” algorithm ($5\% - 25\%$ better performance) and close to the “grd_opt” (less than 10% worse), which does not have any constraints regarding the storage capacity and the total number of installed replicas. This performance is achieved regardless of the size of the network, the capacity of the replication points or the minimum number of replicas installed for each item. The mean hop distance and the network traffic increase sublinearly with the size of the network (Figure 3.4) while increasing the L of every replica the two proposed alternatives and the “rnd” algorithms install more items in “privileged” nodes leading to smaller response delays (and loading with less traffic the network) for every request (Figure 3.5). Moreover, both the mean hop distance and the network traffic decrease as the minimum replication degree (min_k) for each item increases, since now requests reach closer replicas (Figure 3.6).

The “wrr” assignment alternative behaves better than the “seq”, ($4\% - 17\%$ better performance in every conducted experiment), since as explained in Section 3.4.5 this alternative results in a fairer assignment of the items. This means that less popular items still have the chance to be replicated in replication points that match the outcome of Step 1 of the proposed algorithm, leading to better performance for the whole network. As observed by Figures 3.5-3.6, the mean hop distance and the

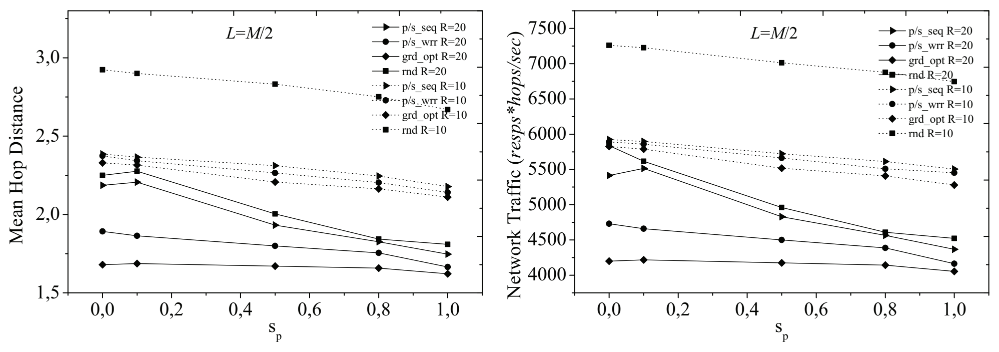

Figure 3.7: Performance of the proposed placement algorithm (both assignment alternatives "seq" and "wrr") compared to the "grd_opt" and the "rnd" vs. the exponent value s_p of the popularity.

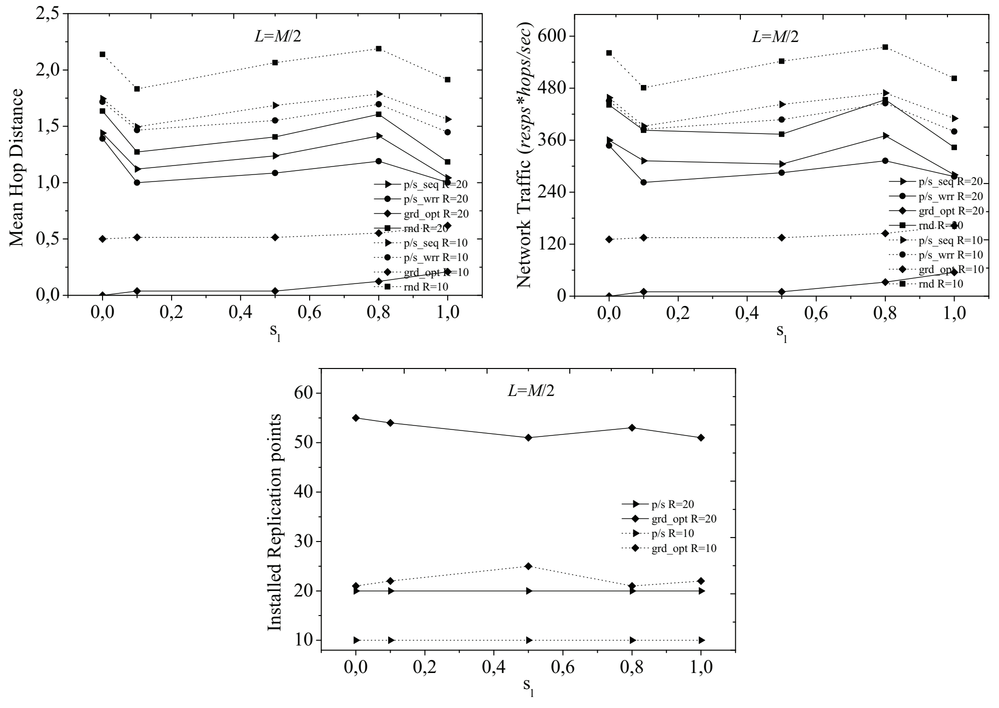

Figure 3.8: Performance and total number of installed replication points in the network of the proposed placement algorithm (both assignment alternatives "seq" and "wrr") compared to the "grd_opt" and the "rnd" vs. the exponent value s_l of the locality.

network traffic graphs have the same form since the storage capacity of each replica and the minimum replication degree of each item does not alter the overall amount of traffic (resps/sec) generated in the network, and the network traffic is the product of the distance (from a client to the closest replica) and the amount of requests generated by the subscribers of the network. Of course, the addition of new nodes (and new subscribers attached to them) increases the amount of traffic in the network and the mean distance between subscribers and replication points; that's why the network traffib graph in Figure 3.4 behaves differently from the mean hop distance graph.

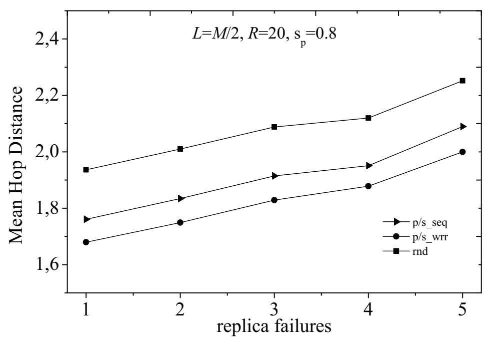

Figure 3.9: Performance of the proposed placement algorithm (both assignment alternatives "seq" and "wrr") compared to the "rnd" vs. the number of storage failures.

Predefined total number of replication points

In this subset, we set two different experiments, one assuming uniform locality and vary the exponent s_p of the popularity and one assuming uniform popularity ($s_p = 0$; uniform popularity means same request rate for each item) and vary the exponent s_l of the locality. Moreover, we assumed that there are $R = 20$ and $R = 10$ available stores ($L = 0.5 \cdot M$ for each replica) that should be placed and assigned (based on the weights) to the items when the network is composed by $V = 100$ nodes.

Figures 3.7 - 3.8 show the mean hop distance and the network traffic for each one of the two experiments. As previously, the proposed algorithm behaves better than the "rnd" algorithm and close to the "grd_opt" when the popularity exponent changes. Particularly the proposed algorithm performs $10\% - 25\%$ better than the random algorithm and less than 9% worse than the greedy optimal algorithm, which has no limitations in the number of installed replicas and their storage constraints. On the other hand, in the experiment where we change the locality exponent, the proposed algorithm performs four times worse than the "grd_opt" but requires three times less replication devices. So, when the offered replication devices are predefined, and the "grd_opt" cannot be used, the proposed algorithm (both the assignment alternatives) performs significantly well.

As previously, the mean hop distance and the network traffic graphs have the same form, since both the popularity and the locality exponent do not alter the overall amount of traffic generated in the network, but only the way that this amount of traffic is allocated among the items and the nodes of the network. For that reason the network traffic metric is not depicted in the following experiments.

We have also conducted an experiment (Figure 3.9; each point is the average of twenty different runs/different combinations of storage failures) assuming storage failures when locality is uniform and the the exponent s_p of the popularity is $s_p = 0.8$. Despite the fact that the proposed planning algorithm and the two assignment alternatives are not designed to take into consideration the possibility of failure of each replication point during their selection, we observe a linear increase in the mean hop distance when storage failures occur. Particularly, we observe up to 11% increase in the mean hop distance when 25% of the replicas fail (fail to serve requests). In the case of a failure, a request will be served from the next closest replication point, so the observed increase in the mean hop distance is the distance between the new replication point and the one that failed.

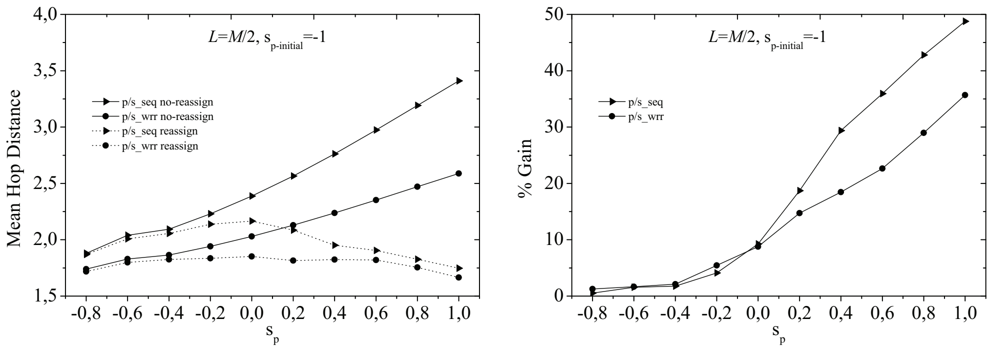

Figure 3.10: Performance and % gain of the assignment phase (both alternatives "seq" and "wrr") of the placement algorithm after an initial planning compared to the placement algorithm without reassignment vs. the evolution of the value of the popularity exponent.

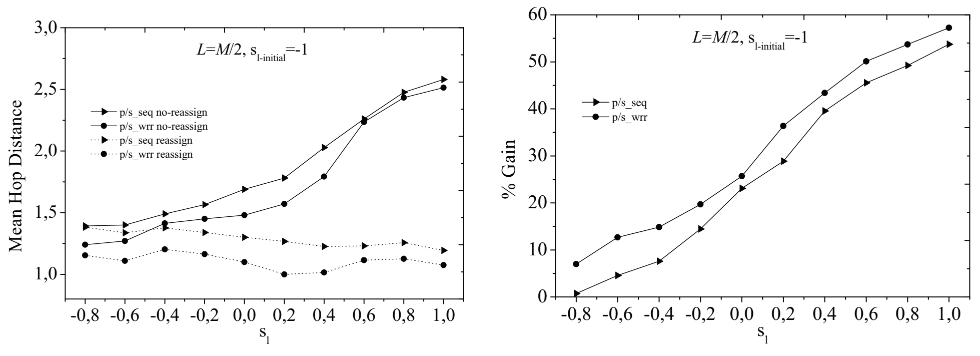

Figure 3.11: Performance and % gain of the assignment phase (both alternatives "seq" and "wrr") of the placement algorithm after an initial planning compared to the placement algorithm without reassignment vs. the evolution of the value of the locality exponent.

3.5.2 Evaluation of the Reassignment Phase

In this set of experiments, we evaluate only the reassignment phase of the proposed algorithm (Step 4 in Section 3.4.3) after an initial planning and assignment of the replicas. Particularly, we run two different experiments. In the first one, we assumed uniform locality and vary the popularity when the initial planning was done assuming $s_p = -1$ (the last item is the most popular). In the second experiment we assumed uniform popularity and vary the locality when the initial planning was done assuming $s_l = -1$ (the last item was requested from the largest neighborhood). Also there are $R = 20$ available replication points ($L = 0.5 \cdot M$ for each replica) that should be placed and assigned (based on the weights) to the items, while the network is composed by $V = 100$ nodes.

Figures 3.10 and 3.11 present the mean hop distance and the relative gain of the reassignment process for the two proposed assignment alternatives. It is obvious that the reassignment of items manages to retain the good performance of the network even if the popularity or the locality pattern changes radically. Particularly, when the patterns of the popularity and the locality are inverted the re-assignment phase by itself delivers up to a 55% decrease in the mean hop distance compared to the case where the assignment of the topics among the replicas do not change after the initial planning. Moreover, the performance of the reassignment phase is less than 8% worse compared to the performance of executing both the planning and the assignment steps after the change of the popularity and the locality pattern (Figures 3.7 and 3.8).

The relative gain graphs of Figures 3.10 and 3.11 could also be used as a benchmark for the storage provider in the decision to reassign or not the items in the replication points of the network upon the detection of a change in the popularity or the locality pattern. Particularly, when the popularity pattern (the exponent value s_p) changes up to 50% from its initial value the reassignment of the topics has less than 10% impact in the decrease of the mean hop distance and the network traffic. This means that a storage provider could skip the reassignment of the items since the initial planning and assignment still performs quite well. On the other hand, when the locality pattern changes more than 25% from its initial value the reassignment phase is necessary since it can decrease both the mean hop distance and the network traffic at least 15%.

3.5.3 Discussion

From the above performance analysis we observe that the performance of the proposed algorithms (planning and the two assignment algorithms) is at any case up to 25% better than the "rnd" even in the cases where the mean hop distance is less than 3 hops. Of course the proposed algorithms behave worse than the "grd_opt" which has no limitations in the number of the installed replicas in the network. Particularly, the proposed algorithms perform very close to the "grd_opt" in the majority of the conducted experiments (1%-15% worse). Only in the case where we change the locality exponent the proposed algorithms behave up to four times worse than the greedy installing on the other hand three times less replicas. This implies that in the real world where a storage provider has limitations in the number of replicas that can install the proposed algorithms is an appropriate solution in almost any scenario.

3.6 Chapter Conclusions

In this chapter, we put forward a new mechanism for replication management in information-centric approaches. The proposed concept equips the CBPS model with the ability to store and retrieve stored information. Moreover, we presented a new placement and replica assignment algorithm that differentiates the information items. Evaluation via simulations of the performance of the system regarding the clients' response latency and the network traffic shows that our placement and replica assignment algorithm is a promising solution in almost any scenario. Finally, the two proposed assignment alternatives could also be used regardless of the initial planning of the replicas to retain good performance of the network when both the popularity and the locality of the requests change. The proposed placement and replica assignment algorithms are generic and can be applied in every information-centric approach.

Chapter 4

Distributed Cache Management and Performance Limits

In this chapter we describe and evaluate the On-line Replacement/Reassignment phase which enables the replacement of information items to the replication points to take place in real-time, based on the changing demand patterns of the users. In contrast to the traditional off-line external management system presented in the previous chapter, here we adopt a distributed autonomic management architecture where management intelligence is placed inside the network.

4.1 Introduction

The proliferation of services deployed over the Internet such as interactive applications, telecommunication services, safety and mission critical systems and also its use for business and social interactions, introduce an increasing need for better quality, dependability, resilience and protection. Current management approaches based on off-line external management systems are inadequate to meet these requirements. As a result, self-management intelligence has to be introduced, within the network in order to make the latter more flexible and adaptive to changing conditions through feedback closed-loop control solutions.

In this chapter, we propose an autonomic cache management architecture for ICN that dynamically assigns information items to caches. Distributed managers make information item (re-)placement decisions, based on the observed item request patterns, such as their popularity and locality, in order to minimize the overall network traffic cost imposed by the user requests. We derive four distributed on-line intra-domain cache management algorithms, categorize them according to the level of co-operation needed and compare them in terms of performance, complexity, message overhead and convergence time. We derive also, a lower bound of the overall network traffic cost for a certain class of network topologies and show that the proposed cache management algorithms perform closely to the derived lower bound.

The rest of the chapter is organized as follows. Section 4.2 presents the functionality of the management substrate that coordinates the caches, while in Section 4.3, we formulate the cache management problem. In Section 4.4, we present the four distributed on-line intra-domain cache management algorithms, while in Section 4.5 the communication and computational complexity of the proposed algorithms is analyzed. In Section 4.6 we derive a lower bound of network traffic cost for various regular network topologies, while in Section 4.7 we evaluate through simulations the performance of the proposed algorithms and we compare their outcome with that of completely selfish techniques. Finally, in Section 4.8 we conclude the chapter.

4.2 Autonomic Cache Management System Architecture

Autonomic self-management envisages systems that can manage themselves given high-level objectives by administrators. Extending autonomic management from individual devices to the collective self-management of networks of such devices results in autonomic networking. In this section, we briefly present a cache management architecture, which given a high-level optimization objective decides in real-time the placement of the items in the caches of the network so as to minimize/maximize the given objective.

Current approaches applied to Content Distribution Networks (CDNs) follow static off-line approaches with algorithms that decide the optimal location of caches and the assignment of information items and their replicas to those caches based on predictions of content requests by users. In contrast, we propose the deployment of an intelligent substrate architecture that enables the assignment of information items to caches to take place in real-time, adapting to the ever-changing user demand patterns. Distributed *Cache Managers* (CMs) decide the items every cache stores by forming a substrate that can be organized either in a hierarchical manner for scalability reasons or in a peer-to-peer organizational structure. The required information such as request rates, popularity/locality of information items and current cache configurations, is exchanged between the distributed cache managers through the intelligent substrate functionality.

Every cache manager (Replacement Component in Figure 2.2), decides in a coordinated manner with other managers whether to cache an item. This may require the replacement of an already stored item (depending on the available space at the cache). The replacements are performed towards maximizing a network-wide utility function (e.g. the gain in network traffic). Thus, every node should calculate the gain the replacement of an item would incur. This approach assumes that every cache manager has a holistic network-wide view of all the cache configurations and relevant request patterns. Such information could be exchanged periodically or in an event-based manner, e.g. when a manager changes the configuration of its cache.

In an alternative approach, managers could base their decisions on a local view of the user demand for specific items but coordinate to maximize the overall network gain, or could even act selfishly towards maximizing their own local utility. Since all the above decisions are made in a distributed manner, uncoordinated decisions could lead to suboptimal and inconsistent configurations. Coordinated decision making of a distributed cache management solution can be achieved through the substrate mechanisms, by ensuring that managers change the configuration of each cache in an iterative manner. In this chapter, we propose and compare several distributed on-line cache management algorithms and evaluate their performance with respect to their autonomicity.

Such an autonomic cache management system can be deployed on top of a PURSUIT-like network architecture following a centralized approach, as well as a distributed approach applied on a NDN-based network. In PURSUIT the entity that decides the caching strategy of the network is the *Cache Manager* (CM) and resides either in a separate management server or it could be co-located with the *Rendezvous Node* (RV). The CM may either take long term cache assignment decisions based on predictions of the item demands or may dynamically control the cache resources based on real-time network information by monitoring the status of the network. The CM would extract the demand patterns by monitoring the RV node, which gathers the subscriptions of every user in the network in order to bind the publishers to the subscribers, and passes the successful bindings to the *Topology Manager* (TM) to compute the paths based on its optimization objectives from the subscribers to the publisher(s) or to one of the replication points. The TM then encodes the path in a Bloom filter and sends it to the selected publisher to start sending the requested information item towards the subscriber. Note also that the functionality of the CM in the PURSUIT architecture can also be realized in a distributed manner as depicted in Figure 2.2 where CMs exist on top of the pre-decided replication points and exchange message through the publish/subscribe model in order to coordinate their item assignment decisions.

In NDN [13] a Cache Manager could be installed in every cache-capable node of the network. Each CM would decide the cache allocation and replacement policy of its node and would monitor the *Pending Interest Table* (PIT) to keep track of the forwarded Interests and to estimate the item demands. Every CM can also configure the cache in the nodes with the appropriate allocation, partitioning and replacement directives and exchanges Interest and Data packets with other CMs to inform them of a new configuration or changes in the demand pattern.

4.3 Problem Formulation

We consider a network of arbitrary topology, represented by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. $\mathcal{V}$ denotes the set of caches and $\mathcal{E}$ the communication links interconnecting them. We use the calligraphic letters to denote sets and the corresponding capitals for cardinality; for example $|\mathcal{V}| = V$.

We denote with $\mathcal{M}$ the set of the M information items available at the network and with s^m the size (in bits) of item m. The information items reside at the caches and requests for content access are generated by the users of the network, with each user being directly connected to a cache node, say its closest one. Cache $v \in \mathcal{V}$ has a storage capacity of C_v bits and serves requests generated with rate $r_v = \{r_v^1, \dots, r_v^M\}$, where r_v^m denotes the aggregate incoming request rate (in requests per second) at cache v for information item m. The r_v vector is an estimation of the actual request pattern based on observed, historical data (within a given time window) and this estimation is used as a forecast for the future behavior of the clients attached at each cache. The optimal way to perform this estimation is out of the scope of this work, but in an ICN implementation this information could be extracted by the cache managers using the method described in Section 4.2. Access requests trigger the transfer of the requested item from a cache hosting the item to the node where the request was generated. A request by node u for an item m cached at node v has a cost equal to the product of the communication cost d_{vu} (e.g. in number of hops, delay, etc) of the path from v to u and the size s^m of the transferred item.

We also denote by $\mathcal{H}$ the set of all possible cache configurations. A configuration $H \in \mathcal{H}$ ($h_v \in \{0,1\}^M$, $\forall v \in \mathcal{V}$) can be represented by a binary matrix of size $V \times M$ and specifies the content of each cache in the network. Actually, h_v^m indicates whether information item m is cached at v.

$$h_v^m = \begin{cases} 1 & \text{if item } m \text{ is cached at node } v, \\ 0 & \text{otherwise.} \end{cases}$$

The following constraints define the set of feasible cache configurations:

$$\sum_{v=1}^{V} h_v^m \geq 1 \qquad \forall m \in \mathcal{M} \tag{4.1}$$

$$\sum_{m=1}^{M} s^m h_v^m \leq C_v \qquad \forall v \in \mathcal{V} \tag{4.2}$$

In particular, the first constraint indicates that each information item has to be stored in at least one cache. Otherwise, the total traffic would become unbounded. The second one captures the fact that each cache has a limited capacity that cannot be exceeded. In a centralized system the objective of the network manager would be to find the system wide configuration H that minimizes the total network traffic cost, $T(H)$.

$$T(H) = \sum_{m=1}^{M} \sum_{\substack{v \in \mathcal{V}: \\ h_v^m = 1}} \sum_{u \in \mathcal{N}_v^m} r_u^m d_{vu} \tag{4.3}$$

where $\mathcal{N}_v^m$ is the set of nodes accessing item m through its replica at cache node v, r_u^m the request rate for information item m generated at node u and d_{vu} is the communication cost (e.g. number of hops, delay, etc) from node v to node u.

Our objective here is to minimize the total traffic cost in the network under the constraints of Eq. (4.1) and (4.2). However, as mentioned in the previous chapter, finding the optimal assignment of the items in the caches, even for a static environment, can be mapped to the Generalized Assignment Problem, which in its simplest form is equivalent to the NP-complete multiple knapsack problem.

An intelligent substrate architecture enables the assignment of information items to caches to take place in real-time, based on the ever-changing user demand patterns. In this study, we assume that the substrate is organized in a peer-to-peer fashion where distributed managers, one responsible for each cache, decide the items that should be stored in each cache according to specific performance criteria. Since there is a one-to-one mapping between cache managers and network caching nodes, we use the same notation $\mathcal{V}$ to denote both. Communication of information related to request rates, popularity/locality of information items and current cache configurations takes place between the distributed cache managers through the intelligent substrate functionality.

4.4 Distributed On-Line Cache Management Algorithms

In this section we present four distributed intra-domain, gradient descent type, on-line cache management algorithms that adapt to popularity and locality changes of the user demands. In this direction, each cache manager may update the contents of its corresponding cache, by fetching new items and replacing existing ones. We call this process item replacement. Nevertheless, the proposed mechanisms differ in the amount of information that needs to be communicated through the substrate, the required level of coordination among the cache managers, and the performance objective. We present them in order of decreasing complexity, regarding the induced computational/communication overhead.

The first one, henceforth called *cooperative*, aims at minimizing the overall network traffic cost and hence each cache manager needs a network-wide view of the request patterns and the current cache configuration. In addition, since each replacement decision affects the whole network, some cooperation in the decision making is required.

The second algorithm, henceforth called *holistic* also aims at minimizing the overall network traffic cost and hence requires the same amount of information. On the other hand, the holistic algorithm requires no coordination of the actions of the cache manager and the required decisions are made autonomously by each manager under the assumption of infinite cost for not retrieving an item (Eq. (4.1)). A variation of this algorithm, that we call *holistic-all*, is also proposed, where all the possible beneficial replacements are performed.

Finally, in the last algorithm, henceforth called *myopic*, each manager needs global knowledge of the items cached in the network, but only local knowledge of demand patterns. Besides, the myopic algorithm assumes that each manager acts autonomously and towards minimizing the traffic related to its own demand.

Throughout the chapter we assume that the underlying content delivery mechanism always directs the requests to the closest cache out of those holding the requested item. Given such an access mechanism in order to minimize the total network traffic, the cache managers have to coordinate their actions towards finding the replication degree and the location where each item should be cached. We also assume that each information item is of unit size ($s^m = s = 1$, $\forall m \in \mathcal{M}$), which is a typical assumption in the literature (e.g. [29], [28]). The proposed algorithms are also applicable in the case of different item sizes as well. However, special care needs to be given, since several items may need to be removed from the cache in order to fit the new item. In this case, the manager could select the items to replace by sorting the already cached items in increasing order of traffic loss per unit of size, similarly to the greedy approximation algorithm of [38].

4.4.1 Cooperative Cache Management Algorithm

In order to deal with the challenging problem of optimal item replication, we propose the following adaptive mechanism. In particular, at each iteration all the cache managers $v \in \mathcal{V}$ execute the following steps in parallel given the current cache configuration H and the corresponding total traffic cost $T(H)$. The proposed algorithm is based on cooperative decision making. At each iteration the cache managers cooperate through appropriate message exchange towards identifying the maximum relative gain. Thus, since any change performed in the cache configuration decreases the overall network traffic cost, the proposed algorithm finally converges to a stationary point where no further improvement is possible. Note that the proposed mechanism does not necessarily converge to the optimal cache assignment, but to a local minimum of the objective function for a given initial cache configuration.

Step 1: Let $\mathcal{S}_v$ denote the set of items that are cached at node v and in at least one more cache in the network. For each item $m \in \mathcal{S}_v$ compute the overall performance loss, $l_v^m = T(\overline{H}_v^m) - T(H) \geq 0$, that will be caused if item m is removed from v, leading to a valid new configuration $\overline{H}_v^m$. In this case all the requests for item m at v will be served by another cache, which is at least that far.

Step 2: Let $\mathcal{P}_v$ denote the set of items that are not cached at v. For each item $m \in \mathcal{P}_v$ compute the overall performance gain $g_v^m = T(H) - T(\underline{H}_v^m) \geq 0$ achieved if item m is inserted at cache v, leading hence to a new configuration $\underline{H}_v^m$. In this case a certain amount of requests for item m will be served by node v, as the closest replica.

Step 3: Each manager v considers as candidate for insertion the item $i \in \mathcal{P}_v$ of maximum performance gain (i.e $i = \arg\max g_v$) and as candidate for replacement the item $k \in \mathcal{S}_v$ of minimum performance loss (i.e. $k = \arg\min l_v$).

Step 4: Each manager v calculates the local maximum relative gain $b_v = g_i - l_k$ and informs the rest of the cache managers through a report message $Rep(b, v, i, k)$.

Step 5: After receiving the Rep messages, each manager calculates the network-wide most beneficial replacement, say $Rep^*(b^*, v^*, i^*, k^*)$, the one of maximum relative gain, and updates its configuration matrix H correspondingly, setting $h_{v^*}^{k^*} = 0$ and $h_{v^*}^{i^*} = 1$. At this point only the configuration matrices of the managers are updated. Once the algorithm has converged, managers fetch and cache new information items and replace cached ones (e.g. fetch item i^* and replace item k^*).

Step 6: Repeat steps 1-5 until no further replacements are beneficial for the network, i.e. no positive relative gain exists.

4.4.2 Holistic Cache Management Algorithm

The holistic algorithm is of similar nature and towards the same objective. Its distinguishing characteristic though is that each manager operates on its own by performing replacements on the respective cache. At each iteration a single cache manager, say $v \in \mathcal{V}$, autonomously decides and executes the following steps. Steps 1-3 are identical to the cooperative algorithm and are omitted:

Step 4: The replacement of maximum relative gain $b = g_i - l_k$ is performed by manager v. The rest of the cache managers are notified through the report message $Rep(b, v, i, k)$.

Step 5: After receiving the Rep message every manager updates its configuration matrix H correspondingly, setting $h_v^k = 0$ and $h_v^i = 1$.

The essence behind the holistic algorithm is that each node performs only valid and beneficial replacements, i.e replacements that lead to feasible cache configurations and improve the overall objective respectively. This process is repeated until a stationary point is reached, where no more beneficial replacements are possible.

Although the cache updates may be applied asynchronously among the nodes, we assume that only a single node may modify the cache configuration at a given time. This is due to the requirement that each manager should know the current cache configuration of the network, in order to calculate the gain and loss metrics. Thus, each modification is advertised to the rest cache managers. Relaxing this assumption would lead to a setting where the nodes make decisions based on outdated information, causing thus some performance degradation and making convergence questionable.

4.4.3 Holistic-all Cache Management Algorithm

In the previously described approaches, a manager performs only the most beneficial replacement at each iteration. However, more than one replacements may be beneficial for the system. Thus, we propose a mechanism where each cache manager performs all the beneficial replacements. In the holistic-all algorithm at each iteration a cache manager $v \in \mathcal{V}$ autonomously decides to update its caching strategy. Due to constraint (Eq. (4.1)) a set of items $\mathcal{I}_v$ that are currently stored only at node v cannot be replaced. Thus, the cache manager has to select, out of the set of candidate items $\mathcal{M} \setminus \mathcal{I}_v$, those ones that minimize the total traffic cost T. Since the selection of an item does not affect the traffic reduction caused by the other items, the best $K = C_v - |\mathcal{I}_v|$ (where C_v the capacity of cache v) can be derived in one shot. The holistic-all algorithm can be thought of as a sequence of Gauss-Seidel iterations [39].

Next we provide an example that motivates the validity of the holistic-all algorithm. Consider the example of Figure 4.1 for two different demand patterns and two different initial cache assignments. In the left scenario the holistic algorithm at the stationary point performs 16% better regarding the overall network traffic cost than the holistic-all algorithm even for a very small network topology of two nodes. On the other hand, in the right part (Scenario 2) for the same network topology, but for different demand pattern and different initial cache assignment the holistic-all algorithm performs

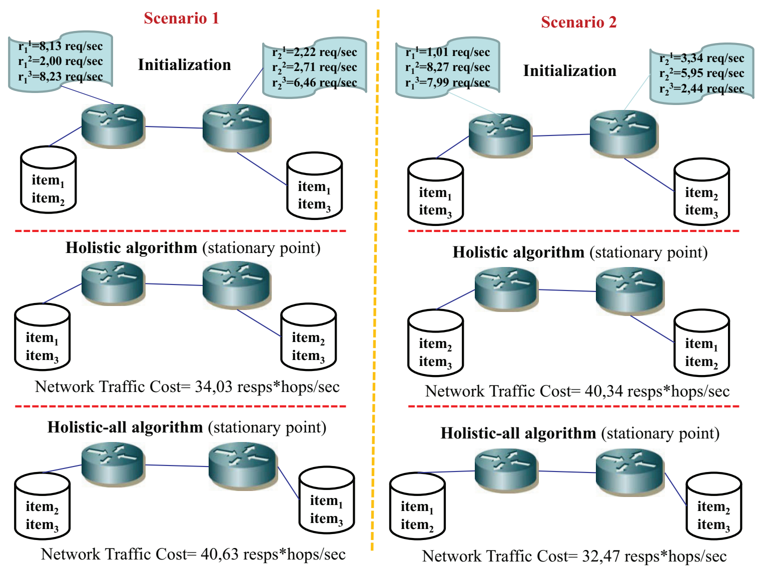

Figure 4.1: A motivation example for two different network and cache configuration scenarios (left side the holistic algorithm performs better than the holistic-all, whereas on the right side the holistic-all algorithm performs better than the holistic algorithm).

20% better than the holistic algorithm. From this simple example is obvious that there are network and cache configurations at which the one algorithm performs better than the other and vice versa, something that motivates the further exploration of both algorithms.

4.4.4 Myopic Cache Management Algorithm

All the previously described algorithms require that cache managers can acquire global knowledge regarding the demand pattern for the decision making. However, in highly dynamic environments the amount of information that needs to be circulated among the managers becomes significant, causing thus non-negligible communication overhead. Even worse the required information may not be available on time, making hence such an approach inapplicable.

For such scenarios we derive an alternative approach. We assume that each manager has no information about the demand patterns at the other caches, and thus makes decisions based only on local information. That is each node v has to select its own cache configuration h_v^m for $m = 1 \dots M$, so as to minimize the traffic cost for the demand it serves. Thus, its objective function now becomes:

$$T_v(H) = \sum_{m=1}^{M} r_v^m d_{u_m v}, \tag{4.4}$$

where u_m is the nearest cache hosting a replica of item m. Although the performance gain and loss expressions used in Steps 1 and 2 of the holistic algorithm have to change according to Eq. (4.4), the main steps of myopic algorithm remains the same with the holistic one. Here, each cache manager tries to minimize the traffic cost of the local demand. Thus, given that the managers do not share a common objective, there might be cases that a replacement at one cache might degrade the objective of another cache. Due to this “competition”, i.e. the counteracting objectives of the individual caches, the myopic algorithm generally requires more iterations and replacements to reach a stationary point.

4.5 Complexity Analysis

In this section we present the communication and computational complexity of the proposed algorithms. This complexity analysis provides insight regarding the incurred computational burden for each cache manager and the communication requirements regarding the substrate, and captures both the initialization and the per iteration complexity. An important parameter that affects total communication and computational complexity of each approach is the number of iterations required for convergence to a stationary point. Since this is difficult to be calculated analytically we perform a characterization through simulations in Section 4.7.

4.5.1 Communication Complexity

We assume that each cache manager is aware of the initial cache configuration of the remote caches. Such information is available at contemporary CDNs [21] and Web caches (digest [40] or summary [41]) and could be easily provided by the ICN implementation, through the Cache Managers.

One of the inherent characteristics of the ICN is the multicast nature of the information dissemination. When a cache manager wishes to disseminate a management message to the rest of the managers in the network this is done through a single transmission over a spanning tree of the network topology (a tree that connects al the nodes/CMs). Such a tree has $V-1$ links, where V is the number of nodes in the network, hence a message from a cache manager to every other manager produces a communication overhead of $V-1$ messages. At the initialization phase of the cooperative, the holistic and the holistic-all algorithm each cache manager needs to have a network-wide knowledge of the demand patterns. This requires each manager to forward its local demand vector to all the other cache managers. For this purpose a message r_v (the demand vector at node v) of size M needs to be forwarded to any other cache manager, leading thus to a total of $V(V-1)$ management messages for the initialization phase. As a result, the communication complexity of the initialization phase is $O(V^2M)$ (assuming that M different messages have to be sent; each message contains the relative information of only one information item). On the other hand, the communication complexity of the myopic algorithm is zero, since decisions are made based only on local demand pattern, information that is available at manager level. Finally, if we assume that the CMs are not always aware of the initial cache configuration of the remote caches, the communication complexity for each one of the proposed algorithms in order to acquire this information is $O(V^2C)$ messages, where C the storage capacity of each node (assuming equal capacity among the caches, $C_v = C$, $\forall v \in \mathcal{V}$). For the rest of the chapter, we assume equal capacity among the caches for ease of presentation.

Regarding the amount of communication overhead induced per iteration per node, in the cooperative, the holistic and the myopic algorithm each manager, in order to calculate the maximum relative

Table 4.1: Communication and Computational complexities of the cache management algorithms

	Complexity		
Algorithm	Commun. for initialization	Commun. per iter. per node	Comput. per iter. per node
Cooperative	$O(V^2M)$	$O(V)$	$O(VM)$
Holistic	$O(V^2M)$	$O(V)$	$O(VM)$
Holistic-all	$O(V^2M)$	$O(VC)$	$O(VM)$
Myopic	0	$O(V)$	$O(M)$

gain, requires a total of $O(V)$ messages. In particular, each manager needs to send its Rep message of length 4 in words to every other manager in the network. Of course, in the cooperative algorithm every manager in the network should send such a message, whereas in the holistic and the myopic only one manager at each time send its Rep message. Thus, the total communication overhead per iteration (total number of management messages in the network) is $O(V^2)$ in the cooperative algorithm and $O(V)$ in the holistic and the myopic algorithms. On the other hand, in the holistic-all algorithm the Rep message that needs to be sent has length of $C+2$, since in the corresponding algorithm each node updates the whole cache every time. This leads to a communication complexity per iteration per node of $O(VC)$ messages.

In order to use an ICN approach for the dissemination of the management messages each cache manager should act both as a subscriber and as a publisher. Particularly, each cache manager should subscribe to the relative management information of every other cache manager (e.g. a given scope in the PURSUIT architecture) and should publish its own management information (e.g. under the same scope in PURSUIT) over the network, so that it could reach the rest of the managers in the network. In other words, the relative management information that is used by the proposed cache management algorithms is treated as another information item by the network.

4.5.2 Computational Complexity

In order to calculate the computational complexity of each algorithm, we define the calculation of the traffic cost for node u to access item m stored at node v (i.e. a single multiplication, $r_u^m d_{vu}$) as a basic operation. Thus, the complexity of each algorithm is calculated as the number of basic operations required.

The cooperative algorithm requires each manager at each iteration to perform $V \cdot M$ basic operations for the calculation of the g and the l vectors. In order to get the g^* and the l^* one max and one min operations are executed by each manager. The max operation has a computational complexity of $O(M-C)$, where C the storage capacity of each node. Similarly, the min operation has a computational complexity of $O(C)$. So the computational complexity of the cooperative algorithm for each iteration per node is $O(VM)$. The holistic algorithm is of the same computational complexity per node. However, here only a single manager computes the relative gain within an iteration and not all of them.

The holistic-all algorithm also requires $V \cdot M$ basic operations per manager for the calculation of the gain of each item. In order to get the K items it requires on average C max operations, where C is the storage capacity of each node. Deriving the C max values out of M items can be performed in $O(M)$, using partial sorting algorithms. So the computational complexity per iteration per node of the holistic-all algorithm is also $O(VM)$.

The myopic algorithm requires for the calculation of the g and the l vectors only M constant time operations per node, since only the local demand pattern is known to each manager. So the computational complexity of the myopic algorithm for each iteration is $O(M)$. Table 4.1 summarizes the communication and computational complexity of the proposed distributed on-line cache management algorithms.

From the above analysis we observe that the complexity of the proposed algorithms, and as a consequence their future applicability, depends on the number of information items M and the number of the nodes V in the network. In this work we assume that the proposed cache management scheme is deployed not at full Internet scale but in a domain scale (like all the current ICN implementations), where the number of items and nodes are significantly smaller and hence our algorithms are applicable. Moreover, the communication and computational complexity could be further reduced using the appropriate aggregation schemes regarding the naming of items, i.e. scopes in PURSUIT and hierarchical names in NDN.

4.6 Network Traffic Lower Bound

As we mentioned earlier, finding the optimal cache assignment is an NP-hard problem. In this section, we focus on the special case of regular network topologies, where all the nodes have the same number of neighbors. The special structure of these networks enables us to derive a lower bound of the overall network traffic cost. Typical examples of such network graphs are the *distance-regular* graphs and the n-dimensional torus graphs.

Here, we consider the case where the communication cost between any two nodes is captured by their hop distance and consequently the total cost is expressed in $responses \cdot hops/sec$. In our setting, in order to maintain the topological equivalency, the demand (request rate) generated from each node v of the network for item m should be the same for all network nodes, i.e. ($r_v^m = r^m$, $\forall v \in \mathcal{V}$).

Since all nodes are topologically equivalent, the cache assignment problem reduces to finding the optimal replication degree $f^m = \sum_{v=1}^{V} h_v^m$ of each item m. For a given replication degree, the best replica assignment is to place the replicas of the same item at equal distance from each other. The exact position of the replicas on the graph does not affect optimality. Thus, such topologies can be "divided" in f^m equal partitions. Each one hosts a replica of item m and serves the same number of neighboring nodes. The above characteristic allows us to characterize the performance limits of those topologies.

For equal demand rates ($r_v^m = r^m$, $\forall v \in \mathcal{V}$), the traffic generated for accessing item m through its replica cached in node v (T_v^m), is given by:

$$T_v^m = r^m \sum_{y \in \mathcal{N}_v^m} d_{yv} \tag{4.5}$$

Let $\ell(x)$ be the function that calculates the sum of the distances of the x nearest neighbors of a cache, according to a Breadth-First Search (BFS). Note that for regular network topologies, the same function holds for every node. The distance from a cache is at least equal to this quantity, i.e.

$$\sum_{y \in \mathcal{N}_v^m} d_{yv} \geq \ell\left(|\mathcal{N}_v^m|\right) \tag{4.6}$$

since the sum of the distances of the nodes ($|\mathcal{N}_v^m|$) accessing item m through its replica at node v cannot be smaller than the sum of the distances of the same number ($|\mathcal{N}_v^m|$) of the closest neighbors of v.

Consequently, for any cache configuration $H \in \mathcal{H}$, using Eq. (4.5) and (4.6) we derive for the overall network traffic cost:

$$T(H) = \sum_{m=1}^{M} \sum_{\substack{v \in \mathcal{V}: \\ h_v^m = 1}} T_v^m \geq \sum_{m=1}^{M} r^m \sum_{\substack{v \in \mathcal{V}: \\ h_v^m = 1}} \ell(|\mathcal{N}_v^m|) \tag{4.7}$$

For the considered network graphs holds:

$$\sum_{\substack{v \in \mathcal{V}: \\ h_v^m = 1}} \ell(|\mathcal{N}_v^m|) \geq f^m \min_{\substack{v \in \mathcal{V}: \\ h_v^m = 1}} \ell(|\mathcal{N}_v^m|) = f^m \ell\left(\frac{V}{f^m}\right), \tag{4.8}$$

since f^m equal partitions of V/f^m nodes are created; the inequality holds due to the convexity of the $\ell(x)$ function. Function $\ell(x)$ is convex since the difference $\ell(x) - \ell(x-1)$ is non decreasing. $\ell(x)$ is derived by $\ell(x-1)$ by the addition of the length of the path from the x-th node to the root (root is the BFS starting point i.e. the cache node under consideration), which is at least equal to the length of the paths of the rest $x-1$ nodes to the root. Hence, node x is at greater or equal distance from the root.

By replacing Eq. (4.8) into Eq. (4.7), we derive:

$$T(H) \geq \sum_{m=1}^{M} r^m f^m \ell\left(\frac{V}{f^m}\right) \tag{4.9}$$

In this setting the optimal replication degree can be derived in polynomial time, as the solution to the following convex optimization problem:

$$B = \min_{f^1,\ldots,f^m} \left(\sum_{m=1}^{M} r^m f^m \ell\left(\frac{V}{f^m}\right) \right) \tag{4.10}$$

$$\text{s.t.} \sum_{m=1}^{M} f^m = V \cdot C \tag{4.11}$$

From Eq. (4.5)-(4.11) we derive the following theorem:

Theorem 4.6.1. *For network topologies that can be represented as regular graphs and all caches are characterized by equal storage capacity ($C_v = C$, $\forall v \in \mathcal{V}$), the solution to Eq. (4.10) is a lower bound of the overall network traffic cost, i.e.*

$$T(H) \geq B,\ \forall H \in \mathcal{H}$$

The following paragraphs are devoted to the calculation of the function $\ell(\cdot)$ for the distance-regular graphs and the n-dimensional torus graphs.

4.6.1 Distance-regular Network Topology

A distance-regular graph is a graph that has no loops or multiple edges, and each vertex has the same number of neighbors. $\mathcal{E}_k(x) \subseteq \mathcal{E}$ denote the set of vertices v with $d_{xv} = k$. For each distance-regular graph there exist integers b_i, c_i, $i = 0, ..., \xi$ such that for any two vertices $x, y \in \mathcal{E}$ of distance $i = d_{xy}$, there are exactly c_i neighbors of y in $\mathcal{E}_{i-1}(x)$ and b_i neighbors of y in $\mathcal{E}_{i+1}(x)$ [42]. A distance-regular graph is characterized by its intersection array:

$$\mathcal{I} = \left\{b_0, b_1, \ldots, b_{\xi-1}; c_1, \ldots, c_\xi\right\}$$

The number of neighbors within a distance of i hops in a distance-regular graph is given by the following recursive formula:

$$h(i) = h(i-1)\left(b_{i-1}/c_i\right),\ 1 \leq i \leq \xi, \tag{4.12}$$

Note that $h(0) = 1$ since each node is a neighbor of itself within zero hops.

From the intersection array of a distance-regular graph and using Eq. (4.12) we derive:

$$\ell(j) = L(i) - (f(i) - j)(i+1) \tag{4.13}$$

where $f(i)$ is the sum of nodes to a given distance i away from a replica, and $L(i)$ is the sum of the distances of those nodes from the same replica and are given respectively by:

$$\begin{aligned} f(i) &= \sum_{p=0}^{i} h(p) \\ L(i) &= \sum_{p=0}^{i} \left(h(p)(p+1)\right). \end{aligned} \tag{4.14}$$

Note that the $+1$ term corresponds to the additional hop to the client.

In order to get i we solve the following inequalities:

$$f(i-1) < j \leq f(i) \tag{4.15}$$

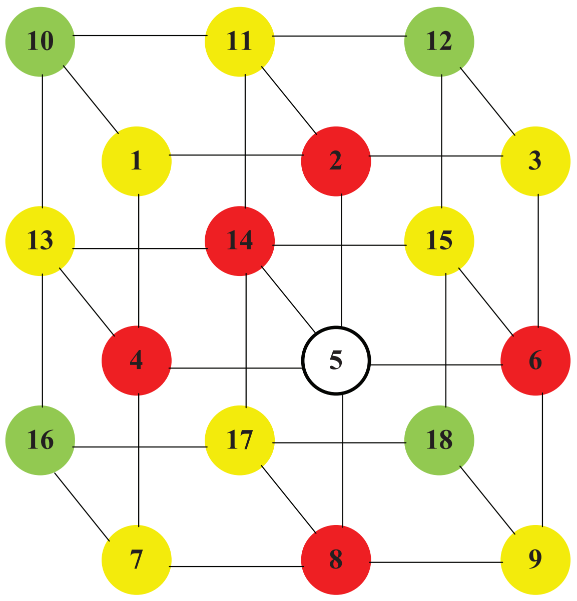

Figure 4.2: A 2-dimensional torus with 18 nodes.

which translates into

$$\sum_{p=0}^{i-1} h(p) < j \leq \sum_{p=0}^{i} h(p) \tag{4.16}$$

When $j = 1$ ($i = 0$) each node has a replica of the given item.

A well known distance-regular topology is the Ring topology. A Ring of length V is a distance regular graph of diameter $\lfloor V/2 \rfloor$. The intersection array of a Ring is $\mathcal{I} = \{2, 1, 1, \ldots, 1, 1; 1, 1, 1 \ldots, 2\}$. In Ring topologies, where nodes are topologically equivalent, the replicas should be assigned on average every $\lfloor V/f^m \rfloor$ nodes. Distance-regular graphs along with their intersection arrays, are presented in [43].

4.6.2 The n-dimensional Torus Network Topology

An n-dimensional torus is an extended torus grid, composed by n different toruses where each node of every separate torus is connected with one node from the other $n-1$ toruses. Thus, the degree of each node is $4 + (n-1)$ (4 the neighbors inside the separate torus and $n-1$ the neighbors of the rest individual toruses), and hence all nodes are topologically equivalent. While in distance regular topologies, the intersection array enables us to calculate the $\ell(j)$, no such array exists for the n-dimensional torus. Here, each node has:

$$h(i) = \begin{cases} 4 + n - 1, & i = 1 \\ 4i + 4(n-1)(i-1), & i > 1 \end{cases} \tag{4.17}$$

neighbors within a distance of i hops ($h(0) = 1$, denotes that each node is a neighbor of itself). For the calculation of $\ell(j)$ we execute the following algorithm:

```
**Require:** $j$: number of nodes accessing the specific replica of the specific item.
**Require:** $n$: dimension of the torus (number of different toruses)
**Require:** $V$: number of nodes in the network topology
  $\ell(j) \leftarrow 1$
  $rm \leftarrow j-1$
  $i \leftarrow 1$ {$i$: number of hops away from the specific replication point}
  $in \leftarrow 1$ {$in$: number of nodes counted from the torus where the replication point belongs to}
  while $rm > 0$ do
    if $i = 1$ then
      $inc \leftarrow min(4, V/n - in)$
      $v \leftarrow \min((inc + (n-1)), m)$
      $in \leftarrow in + inc$
    else
      $inc \leftarrow \min\left(4i, \frac{V}{n} - in\right)$
      $v \leftarrow \min((inc + 4(i-1)(n-1)), m)$
      $in \leftarrow in + inc$
    end if
    $\ell(j) \leftarrow \ell(j) + (i+1) \cdot v$ {+1 the extra hop towards the clients}
    $i \leftarrow i + 1$
    $rm \leftarrow rm - v$
  end while
```

Since, no analytic expression for $\ell(j)$ exists for an n-dimensinal torus, special care has to be given when counting the number of nodes within the same torus (i.e. the torus where the replica is located). For example, we depict in Figure 4.2 a 2-dimensional torus of 18 nodes. Assuming that $j = 18$ (only one replica for the given item placed at node 5) the above algorithms gives 5 nodes in 1 hop away (red), 8 nodes in 2 hops away (yellow), 4 nodes in 3 hops away (green), while using equations similar to Eq. 4.13-4.14 we would have got 5 nodes in 1 hop away, 12 nodes in 2 hops away, 1 node in 3 hops away, which is not true, since the second alternative counts wrongly the nodes within the same torus of the replica point (node 5).

Using an algorithm like the one presented above for those topologies such that no intersection array exists and the analysis of Section 4.6, one could derive a lower bound of the total network traffic cost for any graph consisting of topologically equivalent nodes. Note that the derivation of a lower bound for general topologies may require a completely different methodology and hence it will be considered as future work.

4.7 Performance Evaluation

In this section, we evaluate through simulations the performance of the proposed cache management algorithms. We consider a scenario of $M = |\mathcal{M}| = 1000$ different items, where the request rate for each item at each node is determined by its *popularity*. Here we approximate the popularity of the items by a Zipf law of exponents z_{pop}. Literature provides ample evidence that the file popularity in the Internet follows such a distribution [44]-[47]. We denote by $\vartheta_v = \{\vartheta_v^m : m \in \mathcal{M}, v \in \mathcal{V}\}$ the popularity of each item m at node v.

In particular, we consider seven typical values for z_{pop} (popularity exponents of the Zipf distribution) ranging from -1 to 1, i.e. $z_{pop} \in \mathcal{Z} = \{-1, -0.7, -0.5, 0, 0.5, 0.7, 1\}$). A Zipf distribution of negative exponent (e.g. $z_{pop} = -1$) means that out of the M items the most popular item is the M-th, the second most popular is the $(M-1)$-th and so on, with the first item being the least popular. On the other hand a Zipf distribution of positive exponent (e.g. $z_{pop} = 1$) means that the first item is the most popular and the M-th is the least popular. A zero value of $z_{pop} = 0$ corresponds to equally popular items. We assume that in each node a total of 200 requests per second is generated. Thus, the request rate of each item at each node varies from 0 - 200 req/sec according to its popularity.

Generally, the popularity of each item may differ from place to place, a phenomenon that is referred to as locality of interest. In our experiments, the workload is tuned from a localized subscription model, i.e. similar subscriptions originating from the same region, up to a uniform model. In fact, the locality of similar subscriptions has a significant impact on performance (e.g the efficiency of multicast schemes [48]). Thus, we assume that the network is partitioned in $|\mathcal{Z}|$ neighborhoods. Within each neighborhood the popularity of each item m is constant. We assume that the size of each neighborhood follows a Zipf distribution of exponent z_{loc}, i.e. $\lambda_k : k = 1, \ldots, |Z|$ is the size of partition k, where the popularity of items is given by the corresponding popularity exponent z_{pop}. The impact of locality patterns on the communication-efficiency of several clustering algorithms for content-based routing has also been examined in [49].

In particular, the first partition $k = 1$ consists of $\lfloor \lambda_1 \cdot V \rfloor$ nodes, where the popularity of each item follows a Zipf law of popularity exponent -1. This set of nodes is computed by choosing randomly a central node and its $\lfloor \lambda_\vartheta \cdot V \rfloor - 1$ closest neighbors, by executing a Breadth First Search (as long as a node has not been already assigned to another neighborhood). Note that $z_{loc} = 0$ means that the items are of uniform locality and hence the $|\mathcal{Z}|$ neighborhoods are of equal size ($\frac{V}{|\mathcal{Z}|}$ nodes each). The assumption that locality follows a Zipf distribution is inline with existing literature (e.g. [49] and [48]). However, we show later (Figure 4.7) that the findings of this work are independent of the locality distribution.

The numerical evaluation part consists of two sets of experiments. Initially, we assume uniform locality and popularity (i.e. $z_{pop} = z_{loc} = 0$) for all the information items. Next, we consider scenarios that arise from the synthetic workload generator presented above. In the first set of experiments, we use regular topologies (ring and n-torus) in order to compare the performance of the proposed algorithms to the derived performance bounds. The second set is devoted to the generic case of Internet Topologies from the Zoo dataset [50] and different locality and popularity for each information

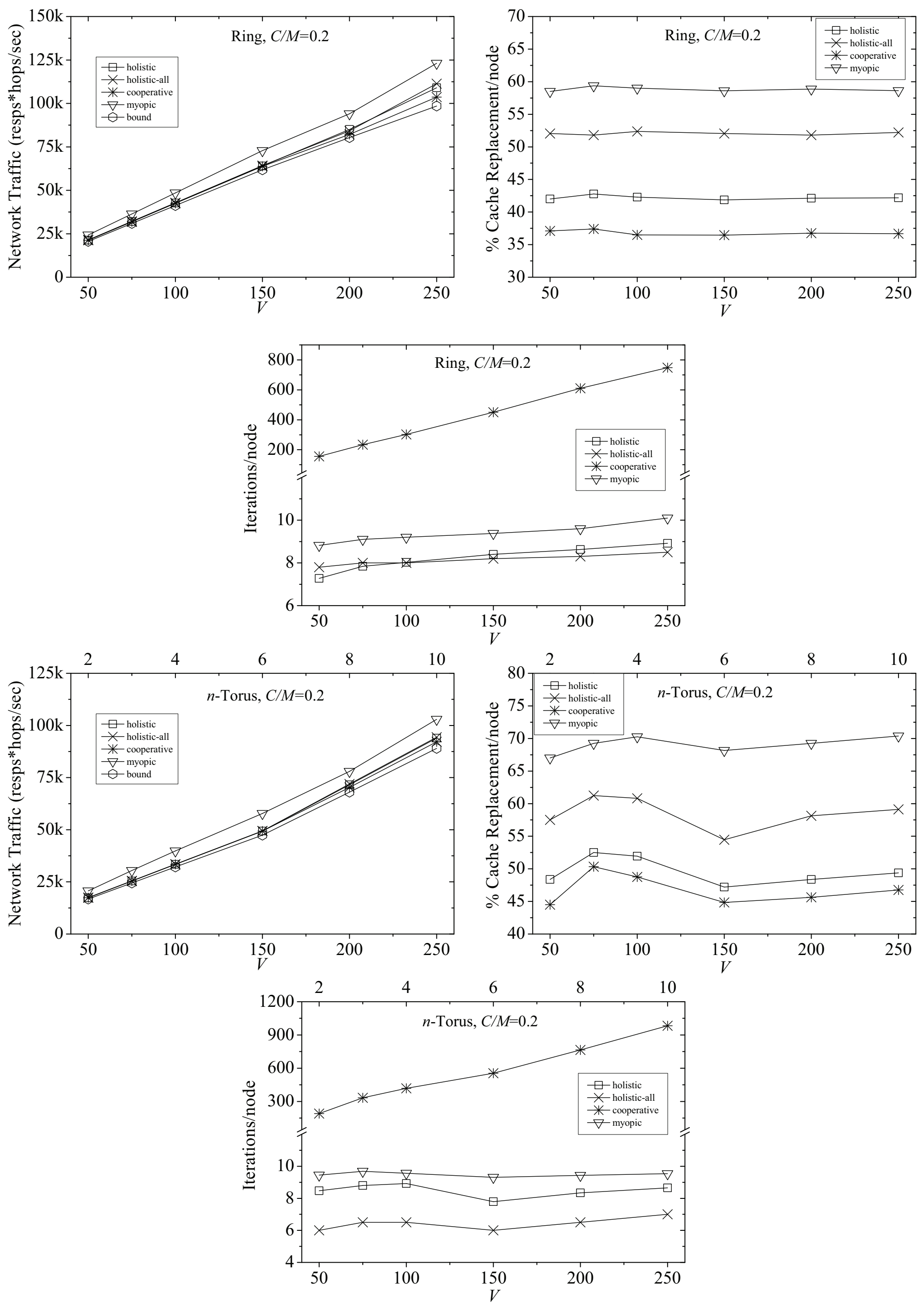

Figure 4.3: The performance of the proposed cache management algorithms vs. the number of nodes/-caches V in the network for two different regular network topologies. The secondary x-axis in the n-Torus plots is the number of different toruses n of the used topology.

item. Indicatively, ring topologies appear in the Zoo dataset, e.g. Hibernia Atlantic (UK), SpiraLight (USA), TelecomSerbia (SRB), Sanren (RSA), etc. Also, regular topologies has also been used extensively in literature, e.g. [51]. In general, we assume that each node of the network hosts a cache and a set of clients is attached to each node and requests items from the network.

Our figures depict for each cache management algorithm the following performance metrics:

- The *overall network traffic cost, NT* (in $resps \cdot hops/sec$) at the stationary point.
- The *replacements, as a percentage of the cache capacity, RE* that have to be performed once the cache management algorithms have converged, i.e. how many items have to be replaced in the cache compared to the initial cache assignment. This is indicative of the communication overhead imposed for fetching the items at the caches.
- The *number of iterations per node, IT* required for convergence, which is indicative of the running time of each algorithm. *IT* multiplied by the communication complexity derived at Section 4.5.1 provides the communication cost of each algorithm per iteration, while multiplied by the computational complexity calculated in Section 4.5.2 reveals the computational cost of each algorithm per iteration.

Since the actual performance of the proposed algorithms depends on the initial cache assignment, the depicted values are averages out of fifty executions, where we start from a random initial cache assignment at each instance.

4.7.1 Performance Evaluation and Bounds for Scenarios of Uniform Demand Patterns and Regular Topologies

Uniform locality and popularity implies that the aggregate request rate generated at each node v of the network for item m is r_v^m and is the same for each node in the network ($r_v^m = r^m, \forall v \in \mathcal{V}$). Figures 4.3 - 4.4 depict the performance of the proposed cache management algorithms for rings and n-dimensional torus network topologies.

Figures 4.3 - 4.4 indicates that the three algorithms that use network-wide information regarding the demand patterns of the users are near optimal in terms of Network Traffic, since the corresponding difference from the lower bound varies between 0.5% and 3.6% regardless of the topology, the size of the network and the storing capacity of each cache. Note that part of this performance gap is due to the continuous relaxation that we use to calculate lower bound B in Eq. (4.10), leading to fractional values in f^m, while in reality only integer values are valid. The three algorithms are on average 3%-15% (depending on the conducted experiment) better than the myopic algorithm. Obviously, as we relax the storage capacity constraint and allow more items to fit in each cache the algorithms present almost identical performance. Moreover, the overall Network Traffic cost increases linearly with the number of nodes in the network V, along with the increase of generated traffic in the network. This implies that the proposed algorithms are not affected by the size of the network.

Regarding complexity, starting from the initial cache assignment the cooperative algorithm requires significantly less replacements to be made, in order to reach the selected cache assignment, than the two holistic approaches as shown by the RE metric. On the other hand in the two holistic approaches each node needs to perform 95% less iterations, as shown by the IT metric. This is the

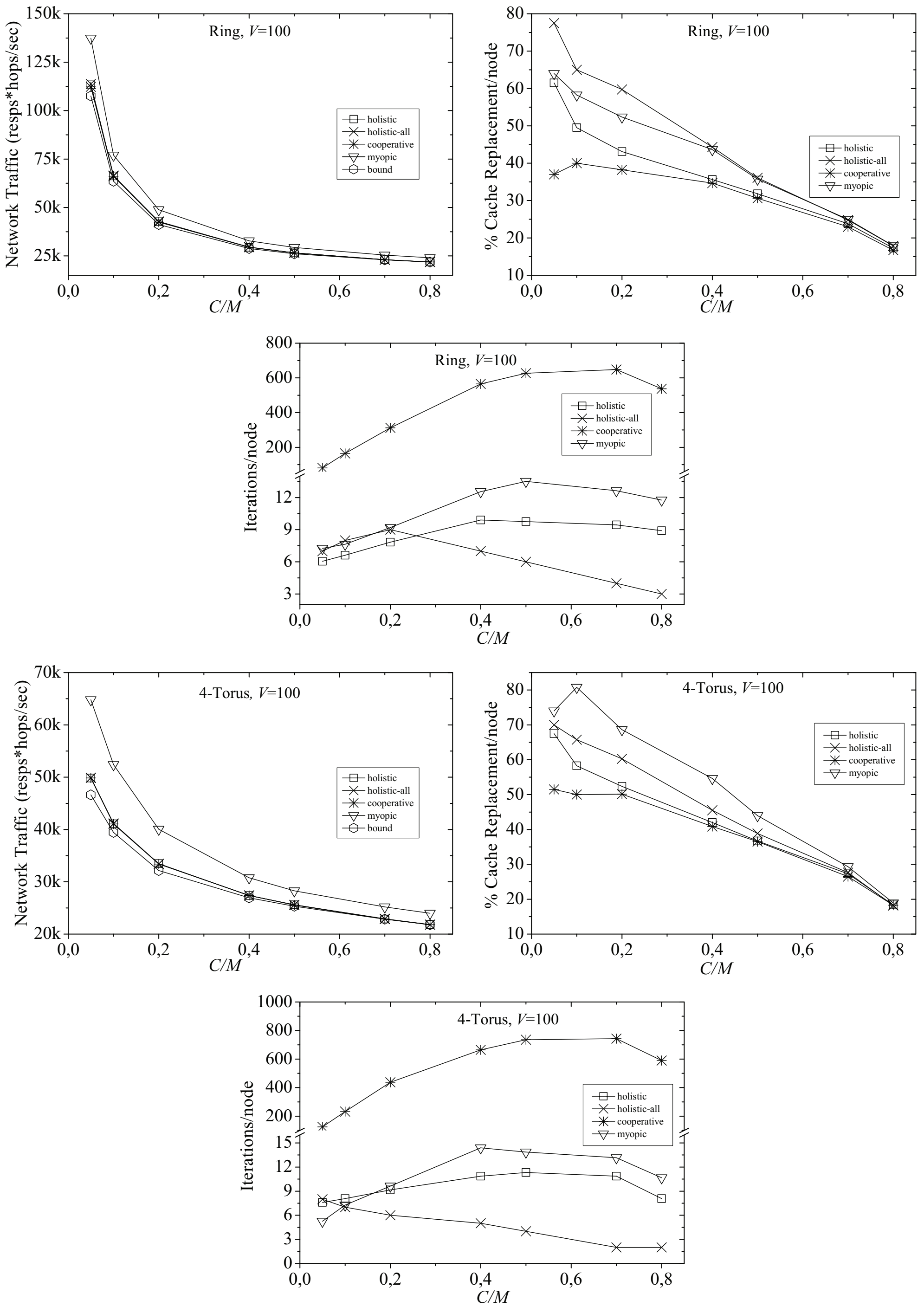

Figure 4.4: The performance of the proposed cache management algorithms vs. the fraction ($p = C/M$) of the items that can be stored in a cache for two different regular network topologies.

cost of coordination, since in the cooperative algorithm at each time all the caches compute a cache update, although in the end only one of them is performed. This also explains the linear behavior of the IT metric of the cooperative algorithm as we increase the number of nodes in the network. On the

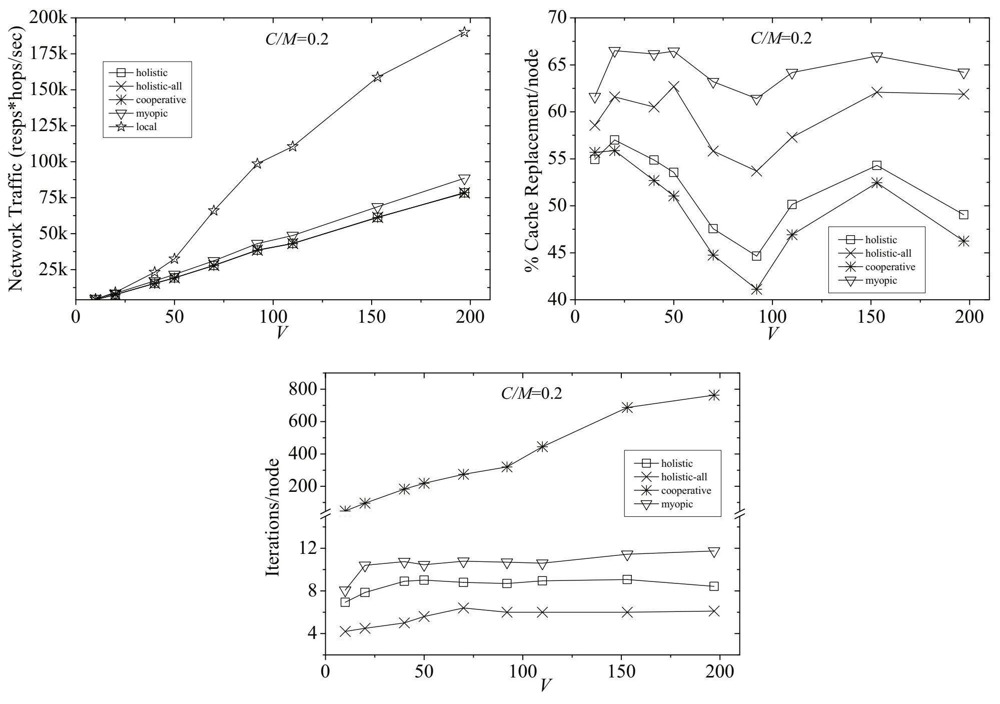

Figure 4.5: The performance of the proposed cache management algorithms vs. the number of nodes/-caches V in the network for various network topologies from the Internet Zoo dataset.

contrary, we observe that the increase of the nodes in the network does not affect the performance of the RE and the IT metric of the two holistic approaches and the myopic algorithm, implying the scalability of uncoordinated decision making, regarding the size of the network. Finally, we observe that the two holistic approaches require $15\% - 20\%$ less iterations per node than the myopic algorithm and this arises from the fact that in the holistic algorithms all the actions are implicitly coordinated through the common objective, while the myopic algorithm suffers from the counteracting objectives of the individuals.

4.7.2 Performance Evaluation for Internet Zoo Topologies and Realistic Synthetic Workloads

The Internet Topology Zoo dataset contains real network topologies from all over the world. Since for a given network size more than one topologies may exist in the dataset, we used all of them for averaging purposes. For comparison purposes, we consider also the *local* selfish approach of [28], in which each manager has no information regarding the remote request patterns and the assignment of the other caches, and thus makes decisions based only on local information. Actually, in such an approach, each cache stores the C locally most popular items. This local selfish algorithm emulates a local LFU replacement scheme, where each cache stores locally the most popular items regardless of the caching decisions of the neighboring caches.

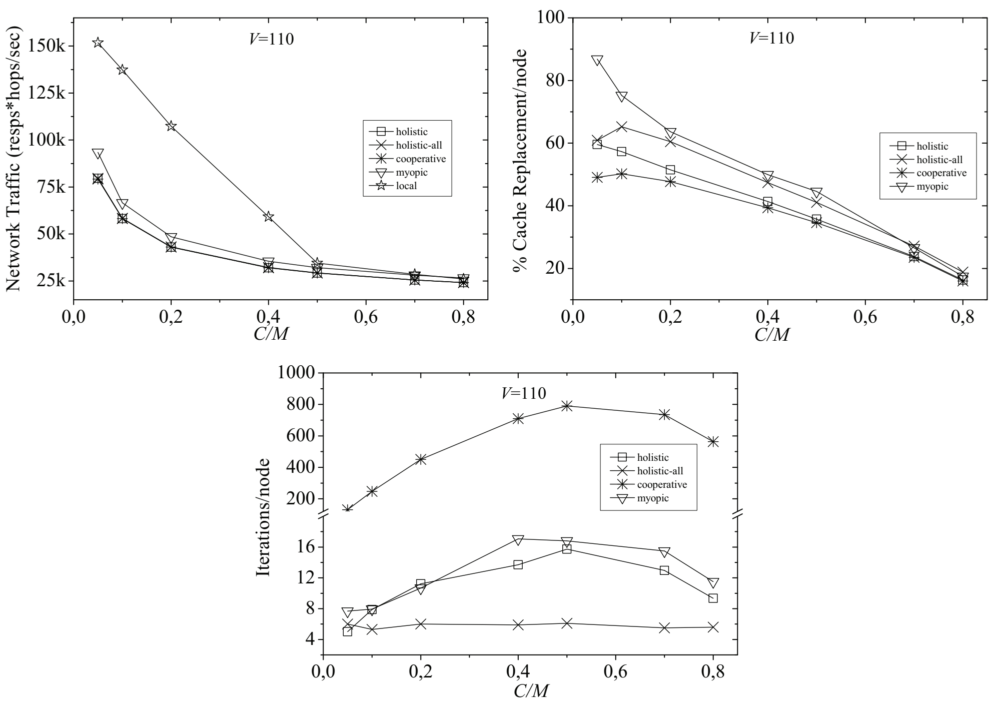

Figure 4.6: The performance of the proposed cache management algorithms vs. the fraction ($p = C/M$) of the items that can be stored in a cache. We used the Interoute network topology from the Internet Zoo dataset.

In this setting, we depict in Figure 4.5 the impact of the number of cache locations V in the network. We notice that the Network Traffic cost metric exhibits linear behavior of slope V similarly to the experiment with the uniform popularity and locality. In addition, the network wide approaches outperform significantly the local one, while they perform 10% better than the myopic algorithm. The cooperative algorithm requires up to 80 times more iterations per node than the other three algorithms but on the other hand requires $5\% - 20\%$ less replacements at the stationary point. Finally, the two holistic algorithms and the myopic are scalable regarding the size of the network since the IT metric is not affected by the number of the nodes in the network.

In Figure 4.6 we depict the impact of the cache capacity on the performance of the proposed algorithms. Regarding the *NT* metric we notice similar behavior with the uniform case, with our network-wide approaches performing better than the local and the myopic algorithm, but as expected with the benefit diminishing as we relax the storage capacity constraint and allowing more items to fit in each cache. However, the complexity related metrics exhibit different behavior. In particular, the *RE* metric decreases almost linearly as the capacity of the caches increase, since the availability of more cache slots enables more items to be stored and hence less replacements are required to reach the selected assignment. Interestingly, the *IT* performance of the holistic-all algorithm is not affected at all by the increase of the cache capacity, while a small increase is observed for the myopic and the holistic one. In contrast the cooperative approach is characterized by an increasing number of

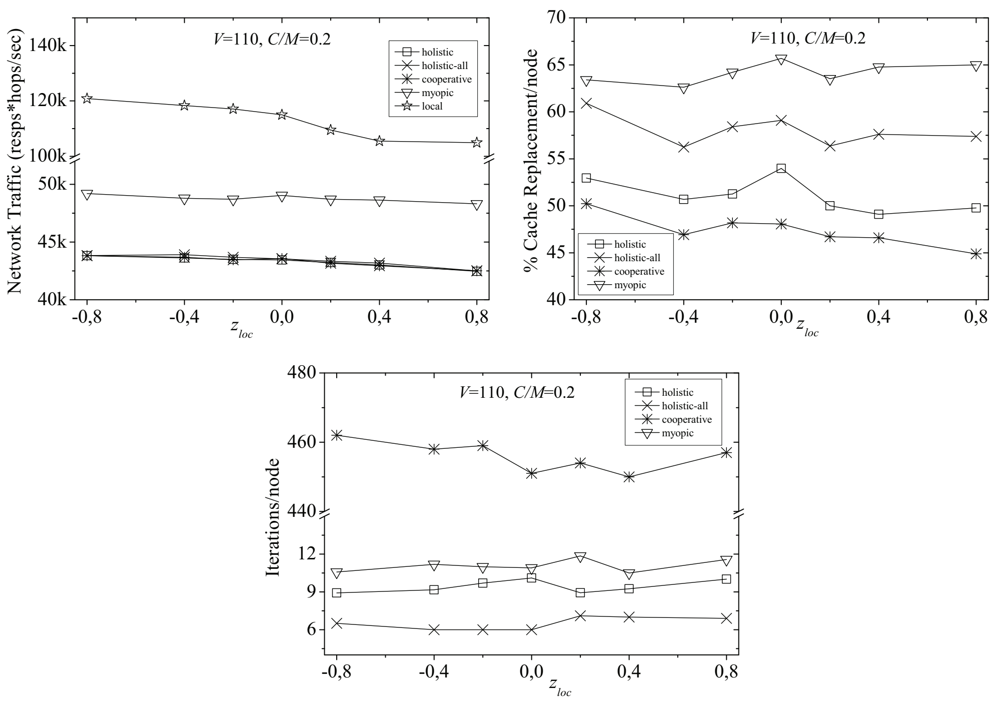

Figure 4.7: The performance of the proposed cache management algorithms vs. the locality exponent z_{loc}. We used the Interoute network topology from the Internet Zoo dataset.

iterations up to $p = C/M = 0.5$ and exhibits a slight decreasing behavior afterwards.

In Figure 4.7 we examine the impact of locality variations on performance. We notice that changes of the locality exponent cause a domino effect requiring significant reorganization of the cache contents, since they alter the topology of the demands of the network under consideration. Of course at the stationary point of operation the algorithms perform identical regarding the traffic regardless of the value of the locality exponents, implying that the sizes of the neighborhoods, where the popularities are assigned, has limited impact on the performance of the proposed algorithms. The minor variations that are observed among different experiments are due to the random selection of the nodes that constitute each neighborhood.

In Figure 4.8 we investigate the adaptability of our algorithms as the popularity of the demand patterns change. Using as initial cache assignment the outcome of an off-line centralized greedy algorithm for given locality and popularity values, we depict the performance of each algorithm as it adapts to the new environmental parameters. The greedy algorithm, reported in [52], is an iterative but off-line centralized algorithm which requires $V \cdot M$ iterations per node. It gives solutions of high quality, since its median performance is within a factor of 1.1 - 1.5 of the optimal and around a factor of 4 for the maximum cases. Particularly, we initially assume that the popularities assigned to the nodes of the network, using a given locality, are given by the vector $Z = (-1, -0.7, -0.5, 0, 0.5, 0.7, 1)$ and at each different experiment (different points in Figure 4.8) this vector changes by a given factor. This factor ranges from 10% to 200%. A change of 10% means

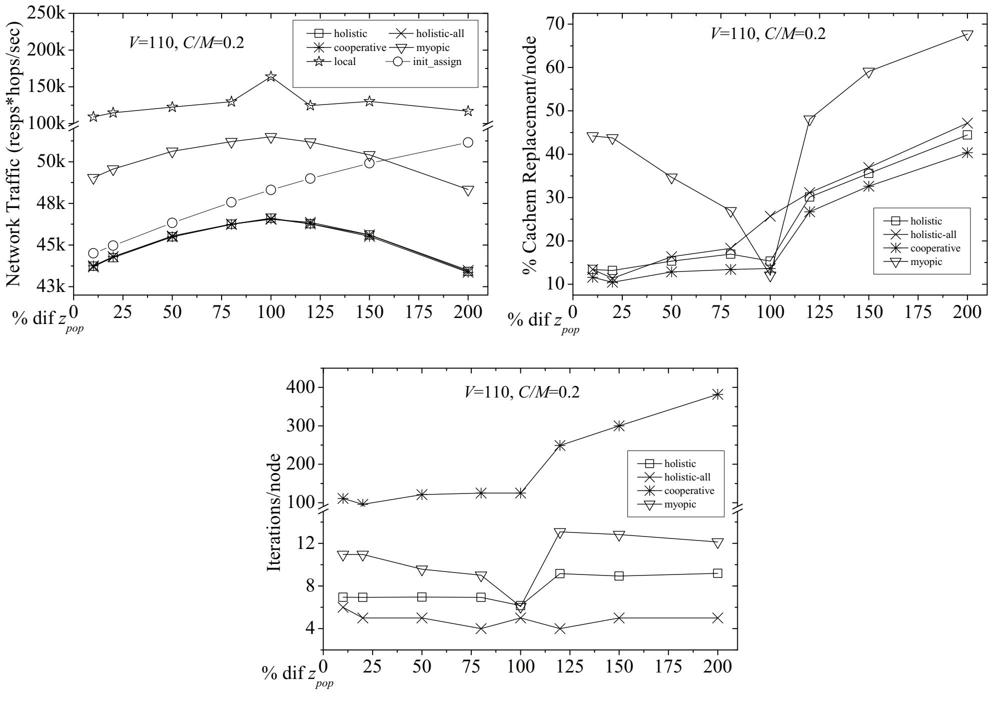

Figure 4.8: The performance of the proposed cache management algorithms vs. the popularity exponent z_{pop}. We used the Interoute network topology from the Internet Zoo dataset.

that the new vector of popularities is $Z = (-0.9, -0.63, -0.45, 0, 0.45, 0.63, 0.9)$, whereas a change of 100% transforms the vector of popularities to $Z = (0,0,0,0,0,0,0)$ and a change of 200% inverts the vector $Z = (1, 0.7, 0.5, 0, -0.5, -0.7, -1)$. We also depict the performance of the initial cache assignment *init_assign* resulting from the greedy algorithm with the new demand pattern.

In Figure 4.8 we observe that as the item popularities become more uniform (near zero exponent) less replacements are required since all the assignments are of almost equal performance. This is also evident from the network traffic plot, where the performance gap of the proposed approaches diminishes. An interesting finding comes from the comparison of the proposed cache management algorithms with the performance of the initial cache assignment. We observe that when the changing factor of the initial popularities is smaller that 100% the algorithms with the network-wide knowledge performs only $1\% - 3\%$ better than the initial assignment and only when the changing factor is larger than 100% and the popularity vector reverts its sign we observe a difference is the performance up to 15% regarding the network traffic cost metric. This means that as long as the ranking of the items, regarding their popularity, does not change and despite the fact that items' popularity becomes more uniform, the initial cache assignment of the off-line greedy algorithm is still good enough and even better than the myopic or the local algorithm.

Note that Figures 4.7 and 4.8 may also serve as a benchmark for the cache managers in their decision to reassign or not the cached items upon the detection of a change in the popularity or the locality pattern. Particularly, the difference between the network traffic cost of the initial cache as-

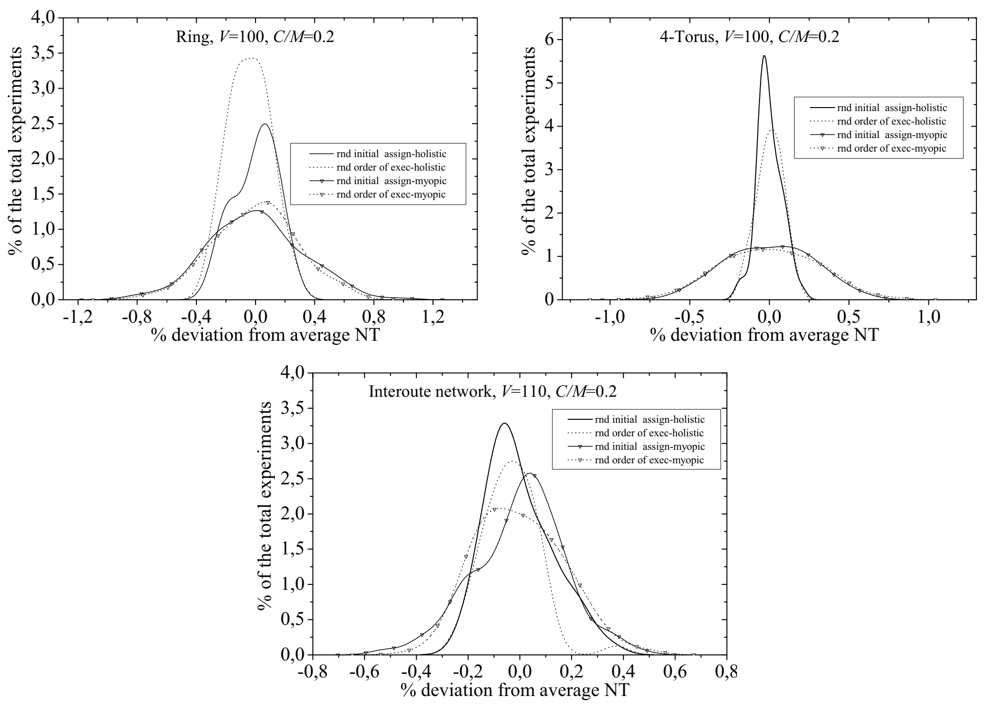

Figure 4.9: Empirical probability density function (pdf) of %-deviation from the average value for the holistic and the myopic approach. The pdfs were generated from 200 different random initial cache assignments and 200 experiments with different random order of execution.

signment and the traffic cost after the completion of the algorithms combined with the communication and computational complexity enables the cache managers to perform or skip the cache reassignment. For example when the observed popularities change up to 75% we observe only a $1\% - 3\%$ decrease on the performance, whereas when the observed popularities revert the decrease in the performance is in the area of 15%, meaning that in the first case the managers could skip the reassignment, whereas in the second case such a reassignment is crucial at almost the same cost with the first case, as shown by the RE and the IT metrics.

Although the performance of the proposed cache management algorithms depends on the initial cache assignment, our extensive simulations indicate that the stationary points are of similar performance. Indicatively, we depict in Figure 4.9 (solid line) the empirical probability density function (pdf) of the %-deviation from the average value for the case of the holistic and the myopic algorithms at the stationary point for a ring, a 4-dimension torus and a topology from the Internet Zoo Topology dataset out of 200 random initial cache assignments. We observe a deviation of up to 0.4% (1.2% in the case of the myopic algorithm) from the average value regardless of the topology. Similar behavior regarding the initial cache assignment has been observed for the rest of the proposed algorithms.

In order to examine the sensitivity of the proposed algorithms to the order according to which the nodes execute the algorithm at each iteration, we also depict in Figure 4.9 (dotted line) the performance (empirical pdf of the %-deviation from the average value) of the holistic and the myopic

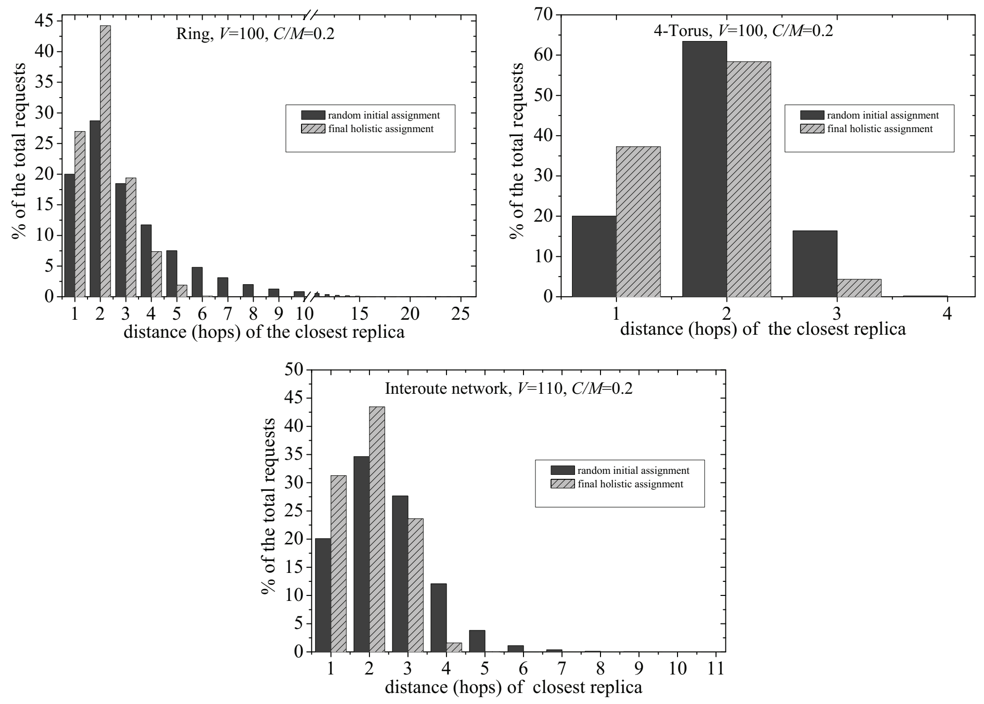

Figure 4.10: The performance of the holistic cache management algorithm (% of the total requests found the requested replica at a given distance), for two regular network topologies and a topology from the Internet Zoo dataset.

algorithms at the stationary point out of 200 different random node order execution experiments, assuming the same initial cache assignment. We observe that the performance of the holistic algorithm deviates only up to 0.3% (1.1% in the case of the myopic algorithm) from the average value regardless of the topology. Interestingly, we observed that the order of execution affects the convergence rate though. In particular, the larger the distance of nodes that execute the algorithm sequentially, the faster the convergence, since a change in a cache affects significantly its neighbors.

In Figure 4.10, we depict the distance (hops) that requests travel in the network until the requested item is found at a cache. The depicted value is the average percentage of requests (out of 200 random initial cache assignments) for a ring, a 4-dimension torus and a topology from the Internet Zoo Topology dataset for the case of the holistic algorithm. We observe that the items are found closer to the requesting node. Particularly, more than 90% of the items are found in less than three hops, whereas before the execution of the algorithm requests should travel at least 50% further. The effectiveness of the algorithm is more impressive in the case of the Ring topology where initially requests should travel up to twenty six hops in order to find the item, whereas the solution of our approach guarantees that 85% of the requests travel no more than three hops and no request travels more than six hops.

Finally, the proposed algorithms are also suitable for the population of new items in the network as well as the replacement of obsolete ones, since new or obsolete items are captured by the evolving popularity (new items will be populated in the caches and old ones will be finally replaced).

4.8 Chapter Conclusions

In this chapter, we proposed an autonomic cache management architecture that dynamically reassigns information items to the caches of an ICN approach. The reassignment decisions are based on real-time information, such as the observed popularity and locality of requests, and not on static off-line predictions. Particularly, we proposed four distributed on-line cache management algorithms requiring different levels of cooperation among the autonomic managers and we compare them in terms of their performance, complexity, message overhead and convergence time. We provided also a method to calculate a lower bound of overall network traffic cost, for distance-regular network topologies. Our numerical results provide evidence that network wide knowledge and cooperation give significant performance benefits and reduce the time to convergence at the cost of additional message exchanges and computational effort.

In particular, the cooperative algorithm provides the best performance regarding overall network traffic, but requires a high level of cooperation among the managers and hence is of very high computational and communication complexity. On the other hand, the two holistic algorithms perform close to the cooperative, but converge in a fraction of the iterations required by the cooperative. In more details, our new holistic-all approach requires significantly less iterations to converge, at a minor performance loss ($< 1\%$) and with slightly more communication overhead than the holistic. Thus, holistic-all fits ideally to highly dynamic environments, where changes in the request rate pattern are very frequent. Finally, the myopic algorithm requires the least cooperation and hence is appropriate for larger network setups, but its performance is significantly worse that the rest. Thus, the analysis of this chapter may serve as a valuable tool for the network manager so as to select the most appropriate algorithm for his needs, depending on specific network parameters (e.g. network size, number of information items, volatility of the request pattern).

Chapter 5

Opportunistic Caching

In an peer-to-peer deployment of an CBPS system, where servers do not exist, information delivery is guaranteed for all active users/subscribers at publish time. However, in a dynamic scenario where users join and leave the network, a user may be interested in information published before the subscription time. In this chapter, we describe and evaluate through simulations and PlanetLab experimentation our design and implementation of an opportunistic caching mechanism. We also propose a stochastic model that captures the dynamics of the proposed mechanism. Each node of the network has a limited cache and we enhance the CBPS architecture with a request/response scheme so that subscribers can retrieve already published information items. Additionally, we present two duplicate preventing mechanisms to deal with the possible production of multiple identical responses. The proposed caching mechanism is compared with traditional opportunistic caching mechanisms with regards to network overhead, delay and mainly information's life-time in the network.

5.1 Introduction

Opportunistic caching on the Web is a thoroughly investigated issue. Particularly in the context of Web caching, NLANR designed the Internet Cache Protocol (ICP) [53] and the HTCP [54] protocol, to support discovery, retrieval and management of documents from neighboring caches as well as parent caches, while in [55] authors discuss and compare the performance of different caching mechanisms, and derive analytical models to study important performance parameters of Web caching, such as clients perceived latency, bandwidth usage, etc. In the area of Delay Tolerant Networks (DTN) in [56] authors studies the performance of caching by the nodes. Generally DTN can provide ad-hoc communication services within (sparse) mobile user communities when end-to-end IP communication is not available. This implies that the nodes cache the data for some time, as DTN operates as a store-carry-and-forward network. The data that is being carried can also be used to serve requests from other nodes before its lifetime has expired.

Recent work on packet caches in routers [57] explores the benefits of deploying packet-level redundant content elimination as a universal primitive on all Internet routers. Moreover, CacheCast [58] is a mechanism to cache data in normal data streams by allowing senders to identify the packet content by a payload ID. Additionally, in [59] authors present CONIC, a network architecture designed for efficient data dissemination using storage and bandwidth resources in end systems. CONIC exploits available storage located in end hosts and uses it as caches. Cached content is then advertised to Content Routers (CR) which maintain a distributed index of cached items based on topological information. CRs track the location of cached objects and if the topological distance to a cache is smaller than the distance to the objects originator the CR sends to the requesting host instructions to explicitly request the object from the caching point.

In the overlay-based caching area, where the data is stored in entities that are specially created for caching purposes, one popular form is web proxy caching [60] - [62] (and the corresponding single-cache system replacement policies [63], [64]), where the ISP places web proxies at strategic locations in the network to cache popular web pages. Hierarchical and cooperative caching, e.g., [65], [66], [67], which has looked into content replication in a string of caches, has assumed some form of cooperation between caches of different levels and hierarchical assignment of caching responsibilities between (a maximum of three) caching levels. Instead, in the case of in-network caching assumed here, the scene is flatter and all caches share equal caching responsibilities.

In general, ICN enables in-network opportunistic caching of information items [68], [5] in every cache-equipped node [69] and replacement of cached items at line-speed [70]. Information items are cached by default in every router that the item traverses and items are replaced using the least recently used (LRU) policy. The cache-everything-everywhere scheme presented in [5] has already raised doubts and some authors have already questioned this aggressive strategy [18]. In that direction in [71] authors instead of caching the same item at every node along the delivery path investigate if caching only in a subset of node(s) along the delivery path can achieve better performance in terms of cache and server hit rates by exploiting the concept of betweenness centrality. Moreover, in [72] authors formulate the caching problem into a Linear Programming problem and propose a novel caching policy named as *Least Benefit*, which takes into account the benefit of a cache hit instead of simply counting the hit number as the Least Frequently Used (LFU) policy does.

In [73] authors propose a new cooperative caching strategy for the CCN architecture that has been designed for the treatment of large video streams with-on demand access. Their aim is to minimize the amount of queries for time-shifted TV that are treated by servers outside the ISP network and the proposed caching strategy manages to halve the cross-domain traffic. In a similar scenario authors in [74] evaluated the performance of a two-layer cache hierarchy under a demand model that reflects a realistic traffic mix. Their results demonstrate that caching Video on Demand in access routers offers a highly favorable bandwidth memory tradeoff but the other types of content that they considered (web, file sharing and user generated content) would likely be more efficiently handled in very large capacity storage devices in the core. Nevertheless, every research attempt regarding in-network caching in ICN, takes as granted the presence of a hosting server for each information item and uses caches in order to improve the delivery of popular content. This implies, that there is a gap in the literature for an efficient opportunistic caching mechanism aiming also at preserving the information over time instead of only making information available in nearer space as in traditional caching schemes.

Such a caching mechanism is useful in peer-to-peer networks wherein each node in the network may act as an independent router and the usage of a permanent storage, or the existence of a server is either not allowed or not profitable to be hosted by a node in the network. A future application among others, that can adopt the information-centric communication scheme and can benefit from such a caching mechanism is a decentralized social network, where users can improve the publication and retrieval of notifications on posted items, such as photos, status updates, message threads, etc. In such a deployment no user is obliged to serve such requests (permanent store or server), but a user can take advantage of similar interests issued by neighboring users and their cached content (buffer) in order to retrieve the required information/data.

In this chapter we:

- Enhance the CBPS architecture with a request/response scheme so that subscribers can retrieve cached information/data from other nodes in the network, assuming that each network node has a limited cache and there are no servers in the network.
- Propose two duplicate preventing mechanisms, that will handle the possible production of multiple identical responses to a request, due to the multiple caching of information/data at different nodes.
- Decompose the caching mechanism in a set of basic policies/strategies, present at each set the most known and traditional policies and propose an information-centric oriented policy at each one of them.
- Propose a stochastic model that captures the dynamics of the newly proposed policies.
- Describe a prototype implementation of the proposed opportunistic caching mechanism, evaluate it through simulations and Planetlab experimentation and compare it against traditional caching mechanisms and mechanisms produced by combinations of the new policies with already known.

The rest of the chapter is organized as follows. In Section 5.2 we present the functionality of the CBPS architecture followed by the description of the proposed request/response scheme. The different policies/strategies are described in Section 5.3 while, in Section 5.4 we propose a stochastic model that captures the dynamics of the proposed caching mechanism. Section 5.5 is devoted to performance evaluation via simulations and in Section 5.6 we describe the implemented prototype and evaluate our system by performing experiments in PlanetLab. Finally in Section 5.7 we propose a modification to the caching mechanism to enable mobility of subscribers while, we conclude the chapter in Section 5.8.

5.2 Enabling Opportunistic Caching

In this section we give a short description of the CBPS architecture and the forwarding mechanism adopted in our work, that is also supported by many popular publicly available implementations

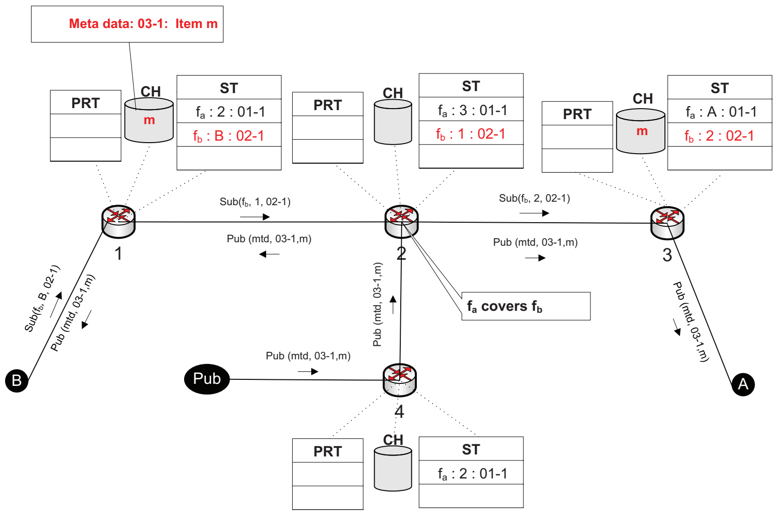

Figure 5.1: Processing of Subscribe and Publish packets.

(e.g. REDS, Siena). We assume an architecture which uses the subscription forwarding routing strategy [15]; the routing paths for the publications are set by the subscriptions, which are propagated throughout the network so as to form a tree that connects the subscribers to all the nodes in the network.

A `Subscribe()` packet contains the corresponding subscription "*filter*" (a list of predicates which define constraints, usually in the form of name-value pairs of properties and basic comparison operators), the "*id/name of the subscriber*" that issued the subscription and a unique "*identifier (sub-id)*" [5] useful to prevent looping the subscriptions. A `Publish()` packet contains an associated "*meta-data field*" describing the published content, also a unique "*identifier (pub-id)*" value and the "*content*" itself. A publication `Pub` matches a subscription `Sub`, whenever the meta-data describing `Pub` matches the `Sub`'s filter.

A subscriber s subscribes by issuing a subscription packet to network node $v \in V$ (V in the number of nodes in the network), that is attached to, (`Subscribe`(f, s, $sub{-}id_s$)) using filter f with a subscription id $sub\text{-}id_s$. Node v inserts the filter, the id of the subscriber and the id of the subscription packet in the Subscription Table (ST$_v$). Then the subscription is broadcasted by v, which now behaves as a subscriber with respect to the rest of the network, to all of its neighboring nodes, with the syntax `Subscribe`(f, v, $sub{-}id_s$). In turn, the neighbors record the subscription and re-propagate it towards all neighboring nodes, except for the one that sent it. Finally, each node $u \in V$ has a ST_u, in which for every neighboring node v there is an associated set of filters $F(v)$ (and subscription identifiers) containing the subscriptions sent by v to u.

Whenever a node receives a subscription packet (from another node) whose id (sub-id) and the corresponding filter are already in its ST, it stops forwarding the subscription without adding an entry in the ST. This procedure is useful to prevent looping subscription packets [5]. Moreover, the subscription scheme is optimized by avoiding subscription forwarding of the same event pattern in

Request packet
Request filter
Subscriber's id/name
Request id (req-id)
Aggregated Pub Id's (APID)

Response packet
Request's id (req-id)
Item's id (pub-id)
Information Item

Figure 5.2: Request and Response packets.

the same direction exploiting "coverage" relations among filters. That is, a subscription is forwarded to a neighboring node only if it is not being covered by a subscription already forwarded to the same neighbor. We say that a subscription sub_i covers another subscription sub_j, denoted by $sub_i \geq sub_j$, iff any event matching sub_j also matches sub_i [75]. Requests to unsubscribe from an event pattern are handled and propagated analogously to subscriptions, although at each hop entries in the ST are removed rather than inserted.

The processing of publication packets is relatively simple. Particularly the node v, attached to the publisher, matches the meta-data of the packet to the filters stored in the ST_v and forwards the packet towards the nodes/subscribers with a matching filter. Finally, all subscribers interested in the published data will receive it. Figure 5.1 depicts the processing of subscription and publication packets.

5.2.1 Caching Mechanism

In this section, we describe the key points of the proposed caching mechanism. We assume that each node is equipped with a limited size cache (equivalent to buffers) and we introduce a request/response scheme in order to provide the network with the ability to make information published in the past available to future clients/subscribers.

Caching points

Each node is selected as a candidate caching point for an item as long as it has in its ST at least one client subscribed in this item. A published information item is transferred to all nodes with client subscribers. Also, a node with a client subscriber is easily reachable by a request packet (the same way it is reachable by a publish packet) through the paths set by the subscriptions. We should clear that only the nodes with attached subscribed clients are candidate caching points and not nodes who have in their STs subscriptions forwarded by other nodes. In that way only a subset of the nodes can be chosen as caching points. Alternative ways to select caching points for a published item are described in Section 5.3.

Request/Response scheme

In order to retrieve cached information, we add to the system two additional types of packets, `Request()` and `Response()` (Figure 5.2). The new packets will allow the traditional CBPS model to acquire

Subscription Table ST

FILTER	SUBSCRIBER'S ID/NAME	SUB ID (sub-id)
artist: Artist Name **type:** mp3	s1 b2	01-001 02-033

Pending Request Table PRT

FILTER	REQUESTOR'S ID/NAME	REQ ID (req-id)
artist: Artist Name **type:** mp3	s2	02-103

Cache CH (buffer)

META-DATA	PUB ID (pub-id)	DATA/CONTENT
artist: Artist Name **song:** song title **album:** album title **type:** mp3	03-222	song.mp3

Figure 5.3: Used Data Structures.

the one-time fetch operation (retrieve content previously published), whereas the publish/subcribe procedure, described above, allows the retrieval of the future content matching a subscription. A `Request()` packet contains the corresponding requesting "*filter*", the "*id of the subscriber*" that issued the request, a unique request "*identifier (req-id)*" and an "*Aggregated Publication Ids (APID)*" variable-length list (APID is used to prevent duplicate responses of the same information item). A `Response()` packet contains the id (or a list as described below) of the responded request "*(req-id)*", the id value of the responded content "*(pub-id)*" and the cached "*content*" itself.

Before describing the request/response scheme, we describe two data structures that we add to the system (at each node): the *PRT* (Pending Request Table) and the *CH* (Cache) (limited buffer memory) (Figure 5.3). The Cache is used to cache published items. The PRT keeps track of requests that have been forwarded towards potential matching caching points so that returned response packets can be sent to its requestor(s).

When a client r, interested in previously published content, appears in the network issues a request by sending a `Request(`$f, r, req\!-\!id_r, null$`)` packet to the node that is attached to (say v; we assume that clients, both publishers and subscribers/requestors are attached only at one node). Node v upon receiving the request packet checks in its Cache for cached items matching filter f (by matching filter f to the cached content meta-data). If a matching item is found, a response packet is initiated and is forwarded back to r. Moreover, node v checks in its ST_v for subscriptions matching filter f. The subscription can be either from another node or from a client. For every existing subscribed node (e.g. u) v forwards the `Request(`$f, v, req\!-\!id_r, \{pub\!-\!id_1, pub\!-\!id_2, \ldots, pub\!-\!id_M\}$`)` ($M$ is the number of matching cached items found in v) packet and inserts an entry in the PRT_v, similar to the one shown in Figure 5.3. If no further node subscriptions exist the request packet is discarded. At a distant node j the syntax of the arriving request, sent by node j', would look like `Request(`$f, j', req\!-\!id_r, \{pub\!-\!id_1, pub\!-\!id_2, \ldots, pub\ -id_X\}$`)`. Each node recipient of a request packet searches in its Cache for information items matching the initial filter f. If a matching item

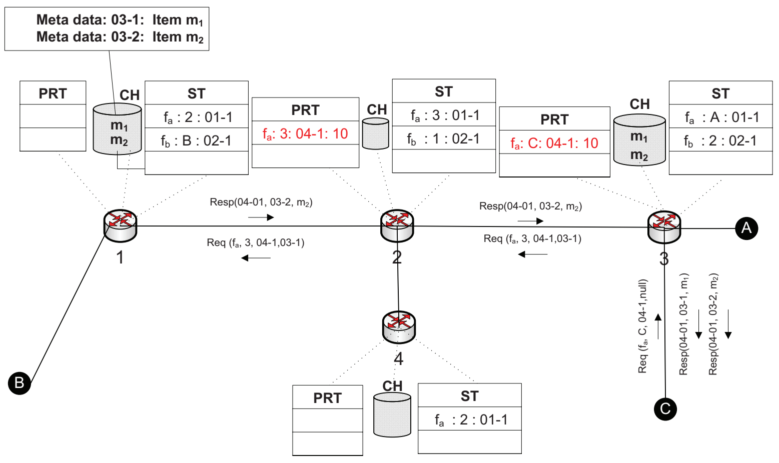

Figure 5.4: Processing of Request and Response packets.

(e.g. m_l) is found and its *pub-id*$_l$ is not in the APID list (*pub-id*$_1$,*pub-id*$_2$,...,*pub-id*$_X$), a response packet is initiated.

The processing of a response packet is relatively simple, since (as in [5] with Data packets) response packets are not routed, they simply follow the chain of PRT entries back to the original requestor(s). Particularly, when a node u receives a response packet, (e.g. `Response(`*req−id*$_r$`,` *pub−id*$_m$`,` *Data−m*`)`) checks in its Cache if the responded item is already cached (using the *pub-id*$_m$) and in case this is true, discards the response packet. Otherwise, checks in the PRT table (in the req-id field) for the *req-id*$_r$. If such an entry does not exist the response packet is discarded, otherwise if the entry's req-id refers to a node (e.g. v) the response packet is forwarded towards that node. If the entry's req-id refers to a client (e.g. s) node u forwards the response packet to the client and caches (or not, depending on the caching policy) the responded item. It is up to the requesting client to sent a message to the network (similar to unsubscribe) to remove its request from the PRTs, otherwise every matching cached information item in the network matching the requesting filter will be sent back to the client.

Handling multiple responses

Multiple caching at different nodes has as side effect the possible production of multiple identical responses on a single request. To deal with this effect, we provide our system with two (one reactive and one proactive) duplicate preventing mechanisms.

In the reactive mechanism, every node, upon the arrival of each response packet, checks whether the responded information item already appears in its Cache and if this is true, it discards the response packet. Otherwise, it forwards the packet according to the technique described above. Responses follow the reverse of the route that the requests follow. This means that the request for initiating the response has also been processed by the node under question which may have responded to that request with the same item(s). Note also, that the requests cannot be dropped in a similar manner, since we consider the CBPS model, and finding a matching item in a proximity node does not guarantee that there are no other (different) items in the network matching the same request.

In the proactive counterpart, every node with a cached matching item, apart from responding to the `Request()` packet appends to the Request's APID list the *pub-id* of the responded item. The nodes –recipients of that request packet– will only respond with items matching the requested filter and their *pub-id* are not in the APID of the request packet, since those information items have already been sent to the client issued that request.

The procedure of responding involves a certain amount of overhead that is unavoidable. Note that the cache of each node is limited and we do not know a priori if the cached items survive the same amount of time at each cache. This means that even if two nodes' STs have clients subscribed for the same filter and have cached in the past the same items (matching that filter) there is no certainty that requesting only one of them will be enough to get those items, since the time that each item "lives" at a cache is not the same and depends on the local workload of each node. Figure 5.4 illustrates the whole request/response scheme and how the mechanisms described here prevent the duplicate response of item m_1.

5.3 Strategies/Policies of the Caching Mechanism

In this section, we present the family of the policies/strategies that are useful to enable the retrieval of cached content in ICN approaches that use the CBPS architectural model.

5.3.1 Caching Policies

A *caching policy* dictates where a published or a responded item will be cached. In principle, all nodes have a cache, nevertheless we may restrict our interest to those nodes having clients interested to the particular item. Requests for content cannot reach a node with no interested clients (unless flooding request is used) since the lack of subscriptions make it invisible to the request/response scheme. Also, it could be possible that every node along the path/route from the publisher/responding node towards the subscriber/requesting client could cache the passing item (similarly to the cache-everything-everywhere scheme used in [5]). We will therefore investigate the following policies:

- Save in all interested nodes – *selective caching policy, SEL.*
- Save in all nodes along the path/route *–en-route caching policy, NRT*.

Selective caching is the newly introduced information-centric oriented policy while the en-route caching is a typical policy found in literature. In the case of a responded item, the selective caching policy caches the item only in the node who hosts the client issuing the request. In the literature more caching policies have been proposed, especially in the ICN area (e.g. in [71] authors investigate caching only in a subset of node(s) exploiting the concept of betweenness centrality) but their application would require additional functionality by the nodes and are not considered in this work.

5.3.2 Placement/Replacement Policies

The *placement/replacement policy* decides a position in the cache where the item will be inserted and which item will be discarded in case of an overflow. We always put a new item (publication or response) at the top of the cache and discard the last in case of an overflow. Then, each newly arrived request may or may not reposition the item to the top of the cache depending on the policy selected. Moreover, the given node may or may not reposition the node at the end of the cache (degradation) when the item under question has lost all the interested clients (all the interested clients have unsubscribed from the node). Finally the traditional LFU (Least-Frequently Used) and LRU (Least-Recently Used) replacement policies will also be examined. Particularly, we will investigate the following policies:

- Put at the top of the cache with regeneration for successive requests and drop the last – *LRU policy*.
- Remove from the cache the item that is used the least and replace it with the new one – *LFU policy*.
- Put at the top of the cache with regeneration for successive requests and degradation when the nodes looses all the interested clients and drop the last – *priority policy, PRT*.

The PRT policy is the newly introduced information-centric oriented placement policy and is a way to differentiate the items in a cache based on their usability, as well as to provide better performance for the whole system. The most popular information items, those with multiple requests are at the top of the cache, while the items with no clients interested in them are at the bottom of the cache, closer to be dropped (since they are not reachable by the requests). We are particularly interested in studying whether this new distributed and intuitive policy can help solve the prioritization issue for items' popularity. The only difference between the PRT and the LRU policies is the degradation of an item when the node looses all the interested clients. In the literature more placement/replacement policies have been proposed [76]-[78] (most of them are variations of the LRU and LFU policies) but their application would require additional functionality by the caches, which might not be applicable for buffer size caches that are assumed here. Finally, analytical models for the efficiency of the traditional LRU and LFU policies have been proposed is various research works [79]-[82] especially for the case of hierarchical caches, but not in the context assumed in this work where the notion of a server is missing.

5.3.3 Request Policies

The *request policy* dictates how the request packet is propagated in the network. According to Section 5.2.1, the request packet is propagated along the subscription tree created by entries in the subscription table that match the requested filter. On the other hand, it is possible to flood the network with requests in order to make sure that any cached item is retrieved at the cost of higher overhead. Particularly, we will investigate the following policies:

- Request based on subscription – *subscription-based request policy, SUB*.
- Request propagated to the whole network – *flooding request policy, FLD*.

The flooding request policy ensures the retrieval of an item (even if it is degraded) but requires the request to visit the whole network, while the newly introduced information-centric oriented subscription-based request policy retains the principles of the used architectural design. Table 5.1 depicts the whole spectrum of the proposed policies, the combination of which result the different opportunistic caching mechanisms.

Table 5.1: Policies (in bold the new information-centric oriented policies).

Policies		
Caching	Plac./Repl.	Request
1. **SEL**	1. **PRT**	1. **SUB**
2. NRT	2. LFU	2. FLD
	3. LRU	

5.3.4 Caching Schemes

From Table 5.1, it is obvious that there exist twelve different combinations of opportunistic caching mechanisms but only a set of those possible combinations is meaningful to be examined. Particularly, we will examine the following five combinations: 1) SEL-LRU-SUB, 2) SEL-LFU-SUB, 3) SEL-PRT-SUB, 4) NRT-LRU-FLD and 5) NRT-LFU-FLD.

The first three combinations constitute our newly proposed caching and request policy, described in Section 5.2.1, combined with the most known placement/replacement policies (LRU and LFU) and the newly introduced priority replacement policy. We compare them with the en-route caching policy combined also with the LRU and LFU placement/replacement policies and the flooding request policy, which is commonly used in the literature. The first three mechanisms are those that maintain the inherent characteristics of the ICN .

5.4 Stochastic Cache Modeling

To the best of our knowledge, there exists no prior work involving analytical models for dynamic information-centric approaches. In this chapter, we assume that subscribers arrive to the network according to a Poisson process, stay in the system for an exponentially distributed time duration and are interested in receiving any information item matching their subscription. Information items that are published in the network arrive according to a Poisson process as well, and the cache equipped network nodes must decide whether to cache them or not. When the caches are full, caching an item means that another item must be dropped. We focus on the survival time of an arbitrary information item, a metric that characterizes the availability of an item in the caches of the network.

We build a multidimensional Markov model which captures the behavior of all the above policies allowing for direct comparison as well as providing intuition for modes of operation. When copying between nodes is not allowed (equivalently when we consider only one node), we find the distribution of information item survival time by analyzing a two dimensional absorbing Markov process. However, even for this simple case the state space is very large and we provide a further approximation to the system reducing the state space significantly, while having a minimal loss in accuracy. For the general problem where copying items from one cache to another is allowed, we propose a heuristic approximation based on estimating the mean rate of copies.

We consider a network with V nodes, each one equipped with a cache. Subscribers arrive in each node requesting content and stay in the system for some time until they disappear. By assuming a potentially infinite population of subscribers, the subscribers arrivals are modeled by a Poisson process. For a given item m and node $i \in V$, the subscribers with subscriptions matching to m arrive with a rate $\lambda_c(m,i) \equiv \lambda_c$ and depart with a rate $\mu_c(m,i) \equiv \mu_c$. Thus we assume that the client dynamics are all the same for all nodes and items (different rates can be used to model item popularity). Consequently, the population sizes of subscribers interested in m are given by $\mathbf{X}_m(t) \doteq (X_{m,1}(t),\dots,X_{m,V}(t))$, where $X_{m,i}(t) \overset{distr}{=} X(t)$ is a birth-death process of the interested subscribers in node i. Given the above rates, the stationary state probability vector for $X(t)$ is $\pi_j = \pi_0 \frac{\rho_c^j}{j!}$, where $j = 0,1,\dots$, $\pi_0 = e^{-\rho_c}$ and $\rho_c = \frac{\lambda_c}{\mu_c}$.

Similarly, we assume that all the new items are published in the system with equal rate λ_b following a Poisson process as well. Items are published only once in the network, survive in the caches of the system for some random time that depends on the caching, request and replacement policies, as well as the subscriber dynamics, arrival rate of the items and the cache size. If an item disappears from all caches at one instance, then clearly it is impossible to be recovered and thus it is lost forever. We then say that the item is *absorbed*.

Let each node be equipped with a cache of size C and consider the process $\mathbf{Y}_m(t) \doteq (Y_{m,1}(t),\dots,Y_{m,V}(t))$, where $Y_{m,i}(t) \in \mathcal{K} = \{0,1,\dots,C\}$ is the position of item m in cache i. Here location 1 corresponds to the top of a cache and the zero point corresponds to the fact that the item is not stored in this cache. Assume that item m is published at $t = 0$ and consider the process Y_m on $t \geq 0$. The state $\mathbf{0} = \{0,\dots,0\} \in \mathcal{K}^V$ corresponds to the absorbing state. Therefore, the item could be lost before stored in any cache. The main performance metric studied is the mean survival time (MST) which is given by

$$\mathrm{MST} \doteq \mathbb{E}\{T_m \,|\, \mathbf{Y}_m(0) \neq \mathbf{0}\}\, \mathbb{P}(\mathbf{Y}_m(0) \neq \mathbf{0})$$

where $T_m = \sup\{t > 0 : \mathbf{Y}_m(t) \neq \mathbf{0}\}$ denotes the survival time of the item m. We also define $P_{\text{loss}} \doteq \mathbb{P}(\mathbf{Y}_m(0) = \mathbf{0})$ as the probability of initial loss of the item.

By Little's law $E(T) = E(M)/\lambda_b$, where M is the number of different items in the system. Thus, increasing the number of different items stored simultaneously increases also MST. It also gives analytical bounds for MST in a homogeneous system. Since the number of different items varies between C (all caches having an identical content) and $V \cdot C$ (all caches storing different items), we have:

$$\frac{C}{\lambda_\mathrm{b}} \leq MST \leq \frac{C \cdot V}{\lambda_\mathrm{b}}. \tag{5.1}$$

In order to get deeper performance results, we need to analyze $(\mathbf{X}_m(t), \mathbf{Y}_m(t))$ in detail. Unfortunately, it is not a Markov process due to the dependence on which items have nonzero subscriber population. On the other hand, the full system $(\mathbf{X}_j(t), \mathbf{Y}_j(t),\ j \in \mathbb{Z})$ is Markovian but too complicated to be studied directly. Thus we will provide approximate approaches in the following sections.

The caching policy determines the initial state probabilities. In `SEL` policy, there is always a probability that the set of nodes, to which interested clients are directly subscribed, is empty. In that case we have a loss event with a probability $P_\mathrm{loss} = \pi_0^V = e^{-\rho_c V}$. The same loss probability also occurs to the `NRT` policy. The caching policy also affects time to absorption since selective caching will reduce the contention at the cache.

The request policy determines the request overhead and the item retrieval efficiency. Particularly, the `FLD` request policy ensures the retrieval of a degraded item but requires the request to visit the whole network, while the subscription based policy (`SUB`) retains the principles of the used architectural design, since the request is propagated towards the nodes with interested subscribers. Another effect of the request policy relates to copying items. Since the flooding request policy always discovers cached items, the copying rate is higher, leading to higher replication degree, less items in the system and smaller MST.

The replacement policy differentiates the survival time of items by bringing popular ones to the top of the cache and thus extending the sojourn time of these items in the transient states. It is then of interest to see whether such an approach is enough to provide a priority mechanism for popular and unpopular items.

5.4.1 The Single-node Case

In this section we consider the simplified network composed of only one node ($V = 1$) which has C cache slots. This model captures also the case where the network consists of more than one nodes, but no item copies from cache to cache are allowed. Then the nodes are behaving independently, i.e., the state probabilities are just products of the single node state probabilities.

First note Eq. (5.1) implies $MST = C/\lambda_\mathrm{b}$, independently of the caching mechanisms. In order to study the distribution and more importantly to prepare the basis for the multi-node ($V > 1$) case, we model both the position of a given item m in the cache as well as the number of items with interested clients (*alive* items), with a two dimensional continuous-time Markov chain (CTMC) with state space $\mathcal{S} = \mathcal{K} \times \mathcal{K}$ and generator matrix $\mathbf{Q}$. The matrix $\mathbf{Q}$ contains the transition rates $q_{\mathbf{s},\hat{\mathbf{s}}}$ from any state $\mathbf{s}$ to any other state $\hat{\mathbf{s}}$, where $\mathbf{s}, \hat{\mathbf{s}} \in \mathcal{S}$, and $\mathbf{s} = \{i, j\}$ is the state where we have i alive items and item m is stored in the j^th slot of the cache. The size of $\mathbf{Q}$ can be reduced to $((C+1)C+1) \times ((C+1)C+1)$ because there are $C(C+1)$ transient states in the chain and we can group all C absorbing states (states of the type $\mathbf{s} = \{i, 0\},\ j \in \mathcal{K}$) into one. Typically, the elements $q_{\mathbf{s},\mathbf{s}}$ of the main diagonal are defined by $q_{\mathbf{s},\mathbf{s}} = -\sum_{\hat{\mathbf{s}} \neq \mathbf{s}} q_{\hat{\mathbf{s}},\mathbf{s}}$.

There are three events that may affect the state of the Markov process, (1) a caching event is triggered by the publication of an item, (2) a regeneration event is triggered by the arrival of a subscriber interested in an item m and (3) a degradation event is triggered by the departure of a subscriber such that the new state for this item becomes $X_m(t+\Delta t)=0$. Accordingly there are seven types of transitions that can take place in system which focuses on a particular item m.

1. The event of caching a new item (other than m) in the given node increases the number of alive items by one and moves item m one cache slot further.

2. The event of regeneration of item m when m was alive retains the number of alive items and moves item m to state one (at the top of the cache).

3. The event of regeneration of item m when m was degraded increases the number of alive items by one and moves item m at the top of the cache.

4. The event of regeneration of an alive item (other than m) retains the number of alive items and may move item m one cache slot further depending on the original position of item m (before or after the regenerated item).

5. The event of regeneration of a degraded item (other than m) increases the number of alive items by one and may move item m one cache slot further depending on the original position of m (before or after the regenerated item).

6. The event of degradation of item m decreases the number of alive items by one and moves m at the bottom of the cache (slot C).

7. The event of degradation of an alive item (other than m) decreases the number of alive items by one and may move m one cache slot towards to the top depending on the original position of m (before or after the degraded item).

In the following we formulate the transition rates according to the above intuitive connection to our system. We define the transition rate from state $\mathbf{s}=\{i,j\}$ to state $\hat{\mathbf{s}}=\{\hat{i},\hat{j}\}$ as

$$q_{\hat{i},\hat{j}}=\lim_{\tau\to 0}\frac{\mathbb{P}(W(t+\tau)=\hat{\mathbf{s}}|W(t)=\mathbf{s})}{\tau},\forall t$$

by omitting the first index of q for presentation reasons. Then for any policy we get:

$$\begin{aligned}
t(1):& q_{0,0} = \lambda_{\mathrm{h}} && j = C, i \le C \\
t(1,4):& q_{C,j+1} = \max\{0, i-j\}\lambda_{\mathrm{g}} + \lambda_{\mathrm{h}} && i = C, j < C \\
t(1,5):& q_{i+1,j+1} = \min\{C-i, C-j\}\lambda_{\mathrm{g}} + \lambda_{\mathrm{h}} && i < C, j < C \\
t(2):& q_{i,1} = \lambda_{\mathrm{g}} && j \le i \le C, 2 \le j \le C \\
t(3):& q_{i+1,1} = \lambda_{\mathrm{g}} && i < j, j \le C \\
t(4):& q_{i,j+1} = \max\{0, i-j\}\lambda_{\mathrm{g}} && i < C, j < C \\
t(5):& q_{i+1,j} = (j-i-1)\lambda_{\mathrm{g}} && i < j, 2 \le j \le C \\
t(6):& q_{i-1,C} = \mu_{\mathrm{d}} && j \le i \le C, j \le C \\
t(7):& q_{i-1,j-1} = \min\{i, j-1\}\mu_{\mathrm{d}} && 2 \le i \le C, 2 \le j \le C \\
t(7):& q_{i-1,j} = (i-j)\mu_{\mathrm{d}} && j < i \le C, 2 \le j < C \\
& q_{\hat{i},\hat{j}} = 0 && \text{otherwise,}
\end{aligned} \tag{5.2}$$

where in general $i, j \in \mathcal{K}$ ($i, j \ge 1$ where not explicitly denoted) and $C \ge 2$, μ_{d} is the rate of item degradation, λ_{g} is the rate at which items are regenerated and λ_{h} is the rate at which item m is *pushed* in the cache by one slot when a new published item is cached in node. Particularly, in the SEL caching policy, items are stored in all nodes with interested subscribers, $\lambda_{\mathrm{h}} = \lambda_{\mathrm{b}}(1-\pi_0)$. For the rate of regeneration we have either $\lambda_{\mathrm{g}} = \lambda_{\mathrm{c}}$ in case the regeneration policy is used, or $\lambda_{\mathrm{g}} = 0$ otherwise. The rate of item degradation is the total rate of transitions of the type $\{X_m(t+\Delta t) = 0 | X_m(t) \ne 0\}$ in the underlying birth-death process $X_m(t)$. Therefore we have $\mu_{\mathrm{d}} = \frac{\lambda_{\mathrm{c}}\pi_0}{1-\pi_0}$.

The studied system is a continuous time Absorbing Markov Process (AMP) (see, e.g., [83],[84]). We renumber the states in the generator matrix so that the transient (the non-absorbing) states come first. So if there are i absorbing states and j transient states, the generator matrix will have the following canonical form:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Tr} & \mathbf{tr} \\ \mathbf{0}^{ij} & \mathbf{0}^{ii} \end{pmatrix},$$

where $\mathbf{Tr}$ is a j-by-j matrix of transition rates among transient states, $\mathbf{tr}$ is a j-by-i matrix of transition rates from transient to absorption states and $\mathbf{0}^{ij}$ a i-by-j matrix of zeros.

Then

$$\mathbb{P}(MST < x) = 1 - \boldsymbol{\phi} e^{\mathbf{Tr}x}\mathbf{e}_j, \qquad MST = -\boldsymbol{\phi}\mathbf{Tr}^{-1}\mathbf{e}_j, \tag{5.3}$$

where $\mathbf{Tr}^{-1}$ is the inverse matrix, $e^{\mathbf{Tr}x}$ is the exponential of the matrix and $\boldsymbol{\phi}$ is the row vector with the initial probabilities of the transient states.

The state probability vector for the transient states $\boldsymbol{\phi} = \{\phi(\mathbf{s})\}$ corresponds to states where item m is positioned at the first slot, i.e., $\mathbf{s} = (i, 1)$, $i = 1, \ldots, k$. Note that the cache is always full with C different items and for the corresponding stationary subscriber birth-death processes it holds $\mathbb{P}(X(t) = 0) = \pi_0$.

$$\phi(\{i, 1\}) = \binom{C-1}{i-1}\pi_0^{C-i}(1-\pi_0)^{i-1}, \quad 1 \le i \le C.$$

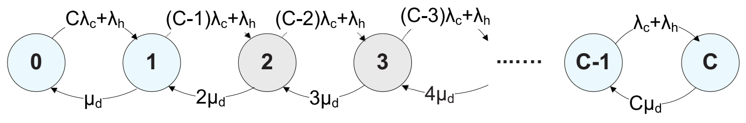

Figure 5.5: CTMC capturing the evolution of the number of alive items in the single node scenario.

The above model can be further applied to other policies for caching, request and replacement as long as the event rates used in the Markovian model are tractable. Finding the distribution for the item sojourn time, i.e., applying Eq. (5.3), requires manipulation of the reduced transition matrix **Tr**. This can be done numerically by any mathematical software after the transition rates in Eq. (5.2) are defined.

5.4.2 Reducing the State Space

A two-dimensional Markov process with $|\mathcal{S}| = C(C+1)+1$ states was developed in Section 5.4.1. However, in order to use this model for a larger network ($V > 1$), it is important to reduce if possible the state space. In this section we propose a further approximation which has a state space of $|\mathcal{S}| = C+1$ states while incurring very small errors in comparison to the original model. The approach is based on stationary analysis of the number of alive items and an AMP which utilizes this stationarity assumption.

In particular, we consider first a stationary CTMC which captures the evolution of the number of alive items in the cache. Figure 5.5 depicts this Markov process and its transitions. The stationary state probabilities are given by:

$$\psi_i = \begin{cases} \dfrac{1}{1+\sum_{n=1}^{C}\left(\frac{\prod_{j=0}^{n}((C-j)\lambda_\mathrm{c}+\lambda_\mathrm{h})}{j!\mu_\mathrm{d}^{j}}\right)} & i=0, \\[2ex] \dfrac{\prod_{j=0}^{l}\left((C-j)\,\lambda_\mathrm{c}+\lambda_\mathrm{h}\right)}{j!\mu_\mathrm{d}^{j}}\psi_0 & 0<i\le C, \end{cases}$$

where ψ_i is the stationary probability of having i alive items in the cache of the node.

We define as U_i the number of alive items in front of item m when m is at slot i. Then assuming independence between the state of item m and the number of alive items, we get:

$$\mathbb{E}\left[U_i\right] = \sum_{j=0}^{C-1}\left(\psi_j \cdot \min\left(j, i-1\right)\right).$$

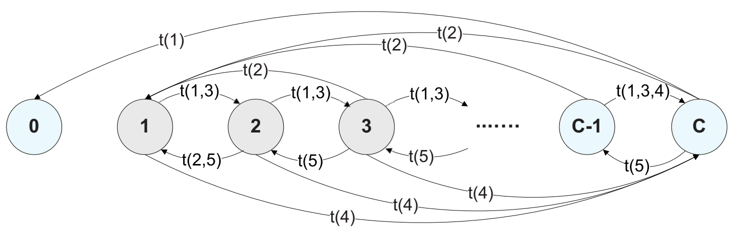

Figure 5.6: Markov chain for the approximated single node scenario.

Moreover, the probability that item m is alive when it is in the i-th slot of the cache is given by:

$$\pi_i^{alive} = \begin{cases} 0, & i = 0, \\ \sum_{j=i}^{C} \psi_j & \text{otherwise.} \end{cases}$$

Using ψ_i, $\mathbb{E}[U_i]$ and π_i^{alive} we can approximate statistically the number of alive items and use the state space $\mathcal{S} = \mathcal{K}$ to denote the position of item m in the cache. In this model, there are five types of transitions that can take place in the chain that tracks the position of item m in the cache.

1. The event of caching a new item (other than m) in the given node moves item m one cache slot further and thus triggers an increase in the state by one.
2. The event of regeneration of item m moves item m to state one (at the top of the cache).
3. The event of regeneration of an item which is before item m (closer to the bottom of the cache) moves m one cache slot further (closer to the end of the cache).
4. The event of degradation of item m (m should be alive) moves item m at the bottom of the cache (slot C).
5. The event of degradation of an alive item in front of item m moves m one cache slot towards to the top of the cache.

For any policy, the transition rates from state i to state j, with $i, j \in \mathcal{K}$, are given by

$$\begin{aligned} t(1) :& q_{C,0} = \lambda_{\mathrm{h}} & \\ t(1,3) :& q_{i,i+1} = (C-i)\cdot\lambda_{\mathrm{g}} + \lambda_{\mathrm{h}} & 0 < i \le C-2 \\ t(1,3,4) :& q_{C-1,C} = \lambda_{\mathrm{g}} + \lambda_{\mathrm{h}} + \pi_{C-1}^{alive}\cdot\mu_{\mathrm{d}} & \\ t(2) :& q_{i,1} = \lambda_{\mathrm{g}} & 2 < i \le C \\ t(2,5) :& q_{2,1} = \mathbb{E}[U_i]\cdot\mu_{\mathrm{d}} + \lambda_{\mathrm{g}} & \\ t(4) :& q_{i,C} = \pi_i^{alive}\cdot\mu_{\mathrm{d}} & 0 < i \le C-2 \\ t(5) :& q_{i,i-1} = \mathbb{E}[U_i]\cdot\mu_{\mathrm{d}} & 2 < i \le C \\ & q_{ij} = 0 & \text{otherwise.} \end{aligned} \tag{5.4}$$

Figure 5.6 shows the transitions of the approximated single node scenario. The initial state probability vector for the transient states is given by:

$$\phi(i) = \begin{cases} \pi_0 & i = 1 \\ 0 & \text{otherwise.} \end{cases}$$

5.4.3 The Multi-node Case

Consider a network composed of V nodes with item copying allowed. In this case, an item can be cached initially in a subset of nodes and later copied to other nodes as well. The item disappears from the network only when it is not cached in any node of the network. We start by making the similar approximation as used for the single node scenario in the previous section. The state space is $\mathcal{S} = \mathcal{K}^V$ and state vector $\mathbf{s} = \{s_1, s_2, \ldots, s_V\} \in \mathcal{S}$ indicates that item m is positioned at the s_i^{th} slot in the cache of node i.

As before, we have caching events that model the publication of an item, regeneration and degradation events of a cached item. In addition to those we should cope with item copies which occur together with the events $\{X(t+\Delta t) = 1 | X(t) = 0\}$. We assume that all events concerning subscriber's activity affect only one dimension of the state, i.e., when an item is requested and sent, it will not be cached in the cache of any intermediate node other than the serving one and only the newly arrived subscriber will receive it. Thus, for each transition from state $\mathbf{s}$ to state $\hat{\mathbf{s}}$, we get $\hat{s}_i = s_i$, $\forall i \neq j$, where j is the dimension where the change is taking place.

For events which depend only on subscriber activities, we write $q_{\mathbf{s},\hat{\mathbf{s}}}(j) = q_{s_j,\hat{s}_j}$ where, $q_{s_j,\hat{s}_j}$ is calculated using Eq. (5.4). We collect these transitions in matrix $\mathbf{Q}_1$. However, a special care is required when $s_j = 0$. In such cases, an item may be copied from another cache reachable by the chosen request policy. Thus we get:

$$q_{s_j=0,\hat{s}_j=1} = f^{\text{request}}(\lambda_{\text{c}}),$$

where $\mathbf{s} \neq \mathbf{0}$ and $f^{\text{request}}(\lambda_{\text{c}})$ depends on the request policy used and the prioritization or not of the cached item.

- For the flooding request policy we get $f^{\text{request}}(\lambda_{\text{c}}) = \lambda_{\text{c}}$
- For the subscription-based request policy we get

$$f^{\text{request}}(\lambda_{\text{c}}) = \left(1 - \prod_{k=1}^{V-1}(1 - \pi_{s_k}^{\text{alive}})\mathbb{1}_{\{s_k>0\}}\right)\lambda_{\text{c}}.$$

The rest of the transitions in $\mathbf{Q}_1$ take place independently in each dimension and they are given by Eq. (5.4) with the exception of the transitions reflecting the item copies. We call this additional pushing rate as $\lambda_{\text{cp}}^{\text{p}}$ (p indicates that this rate is policy-dependent).

Next we deal with events that reflect the publication of an item other than m. These events can cause a change in multiple dimensions of the state space. Specifically, in each node where item m is cached, a new item may appear and cause a push of item m. The caching event in this case depends on the caching policy. In the following, the transitions in case of selective caching policy are shown. Consider the set $\breve{\mathcal{S}} = \{\mathbf{s}_2 \in \mathcal{S} : \mathbf{s}_2 = \mathbf{s}_1 + \mathbf{u}, \mathbf{s}_1 \in \mathcal{S} \setminus \{\mathbf{0}\}, \mathbf{u} \in \{0,1\}^V\}$. For all $\mathbf{s} \in \mathcal{S} \setminus \{\mathbf{0}\}$ and $\hat{\mathbf{s}} \in \breve{\mathcal{S}}$ we write:

$$q_{\mathbf{s},\hat{\mathbf{s}}} = (1-\pi_0)\lambda_{\mathrm{b}}.$$

By collecting the above transition rates in $\mathbf{Q}_2$, we finally obtain the generator matrix as $\mathbf{Q} = \mathbf{Q}_1 + \mathbf{Q}_2$.

The information required to compute the rate of item copies $\lambda_{\mathrm{cp}}^{\mathrm{p}}$ is the random process $M_i(t)$ defined as the number of items in the network that differ from those cached in node i. Process $M_i(t)$ is hidden from our Markov chain and thus we need to rely on an approximation. Process $M_i(t)$ takes values in $[0,(V-1)C]$ with the minimum attained when all caches contain exactly the same items and the maximum attained when all items are cached exactly once. Note that $M_i(t)$ is not stationary and its behavior depends greatly on the ratio of $\lambda_{\mathrm{c}}/\lambda_{\mathrm{b}}$.

In this thesis we make a gross approximation of $M_i(t)$. In particular, we assume that at each time epoch all items in the network have identical replication pattern as that of item m. On the average, this is, of course, true because of i.i.d. subscriber dynamics and publication arrivals. Let $R_m(\mathbf{s})$ be the number of copies of m cached in the network, i.e., it counts the total number of non-zero elements in $\mathbf{s}$ and thus is state dependent but known at each state. We make the approximation that $\tilde{M}(\mathbf{s}) = \left\lfloor C\left(\frac{V}{R_m(\mathbf{s})} - 1\right)\right\rfloor$. The intuition behind this approximation is driven by simulations where we observed that the number of replicas in the network has very small variance and has quasi-stationary behavior soon after the publication of each item.

Then depending on the request policy we have:

- For the flooding request policy we get $\lambda_{\mathrm{cp}}^{\mathrm{FLD}}(\mathbf{s}) = \tilde{M}(\mathbf{s}) \cdot \lambda_{\mathrm{c}}$.
- For the subscription-based request policy we should calculate how many out of the $\tilde{M}(\mathbf{s})$ items could be copied. Defining $p_{\mathrm{fail}}(\mathbf{s}) = \pi_0^{R_m(\mathbf{s})}$ as the probability of an item not to be copied, we get $\lambda_{\mathrm{cp}}^{\mathrm{SUB}}(\mathbf{s}) = \tilde{M}(\mathbf{s})(1-p_{\mathrm{fail}}(\mathbf{s}))\lambda_{\mathrm{c}}$.

The transitions from (5.4) in $\mathbf{Q}_1$ that are affected from $\lambda_{\mathrm{cp}}^{\mathrm{p}}$ and their new form are:

$$\begin{aligned} t(1) &: q_{C,\mathbf{0}} = \lambda_{\mathrm{cp}}^{\mathrm{p}}(\mathbf{s}) \\ t(1,3) &: q_{i,i+1}(\mathbf{s}) = (C-i)\cdot\lambda_{\mathrm{g}} + \lambda_{\mathrm{cp}}^{\mathrm{p}}(\mathbf{s}), \qquad 0 < i \le C-2, \\ t(1,3,4) &: q_{C-1,C}(\mathbf{s}) = \lambda_{\mathrm{g}} + \pi_{C-1}^{alive}\cdot\mu_{\mathrm{d}} + \lambda_{\mathrm{cp}}^{\mathrm{p}}(\mathbf{s}) \end{aligned} \tag{5.5}$$

Also, the initial probability vector for the transient states, i.e., $\mathbf{s} \neq \mathbf{0}$, is given by:

$$\phi(\mathbf{s}) = \begin{cases} (1-\pi_0)^{\sum_i s_i}\pi_0^{V-\sum_i s_i}, & s_i = 0,1, \forall i \in \{1,\ldots,V\} \\ 0, & \text{otherwise.} \end{cases}$$

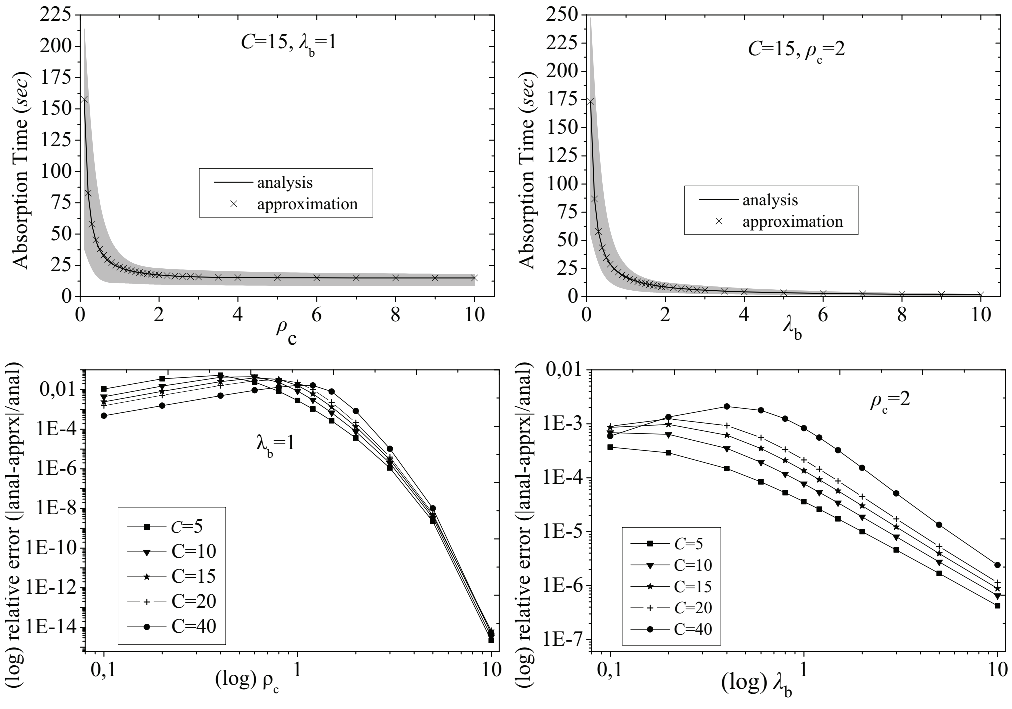

Figure 5.7: Single node scenario: (up) The gray area represents the 50% of data–data between the 1st and 3rd quartile. (down) relative error between analysis and approximation for several parameter settings.

5.5 Performance Evaluation

In this section, we evaluate the proposed opportunistic caching mechanism using a discrete event simulator as well as the validity of the proposed stochastic model. We assign the popularities to the information items from a Zipf-like distribution (parameter equal to 0.7). Each information item survives in the caches of the system for some random time that depends on the policies described in Section 5.3, as well as the subscriber dynamics, arrival rate of the publications and the cache size.

Initially we present results from the proposed models set side by side with discrete event simulations. The goal is to validate the models, show that the proposed approximations yield accurate results. The metric for comparison is chosen to be the mean absorption time (AT), which is the time from the publication of an item until it disappears from the network. The larger the AT the better, indicating that an item survives longer in the system cache and thus it is available for longer time.

5.5.1 Validation of the Single-node Model

In order to validate our models we examine the `SEL-PRT-SUB` caching scheme. Apart from AT, we are also interested in the relative error between the original model (we call it analysis) and the one with the reduced state space (we call it approximation). The AT and the relative error are both random variables and we estimate their mean by simulating 50k of observations. We set out two experiments, one varying the client intensity ρ_c ($C = 15$ and $\lambda_b = 1$) and one varying the rate of newly published

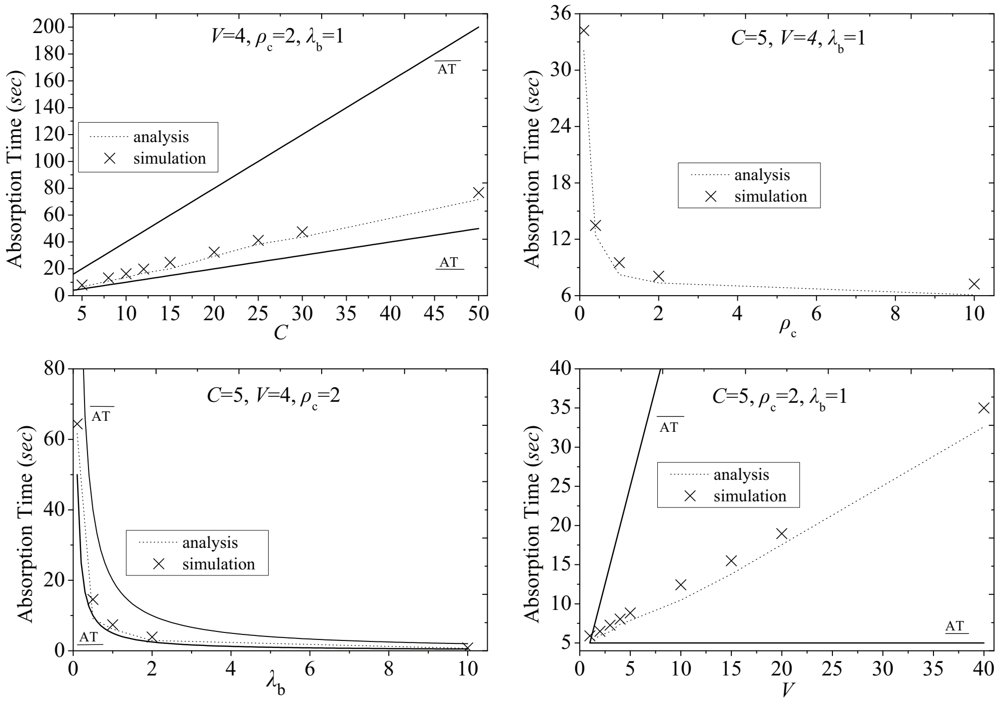

Figure 5.8: Multi-node: AT vs C, ρ_c, λ_b, and V.

items λ_b ($C = 15$ and $\rho_c = 2$).

The gray area in Figure 5.7 represents the area between the 1st and 3rd quartile and thus the 50% of the mass of data. From these Figures we extract the conclusion that the proposed models predict very accurately the AT for several settings. Also, the approximation brings a very small error.

The relative error between analysis and approximation is also showcased for several scenarios in Figure 5.7. The relative error is always smaller than 1% and in many cases is lower than 10^{-4}. This implies that approximating the number of alive items is well motivated and worthwhile.

5.5.2 Validation of the Multi-node Model

In this section we validate the proposed model for the multi-node scenario using the discrete event simulator. For validation reasons we consider the `SEL-PRT-SUB` caching scheme as before. We set out four experiments, one varying the number of cache slots C, one varying the subscriber intensity per node ρ_c, one varying the rate of new publications λ_b and one varying V, the number of nodes in the network. We define as $\overline{\mathrm{AT}}$ and $\underline{\mathrm{AT}}$ the upper and lower bound respectively obtained from Eq. (5.1).

Figure 5.8 depicts AT for several experiments. Note that AT decreases fast with λ_b and ρ_c and increases linearly with V and C. Also it is notable that despite the approximations that we have used, the model is close to simulation having an error of at most 10% in the shown cases. The error seems to increase when V and C increase where the assumption for uniform replication is less accurate. On the other hand, the accuracy seems to be rather invariant to λ_b, ρ_c.

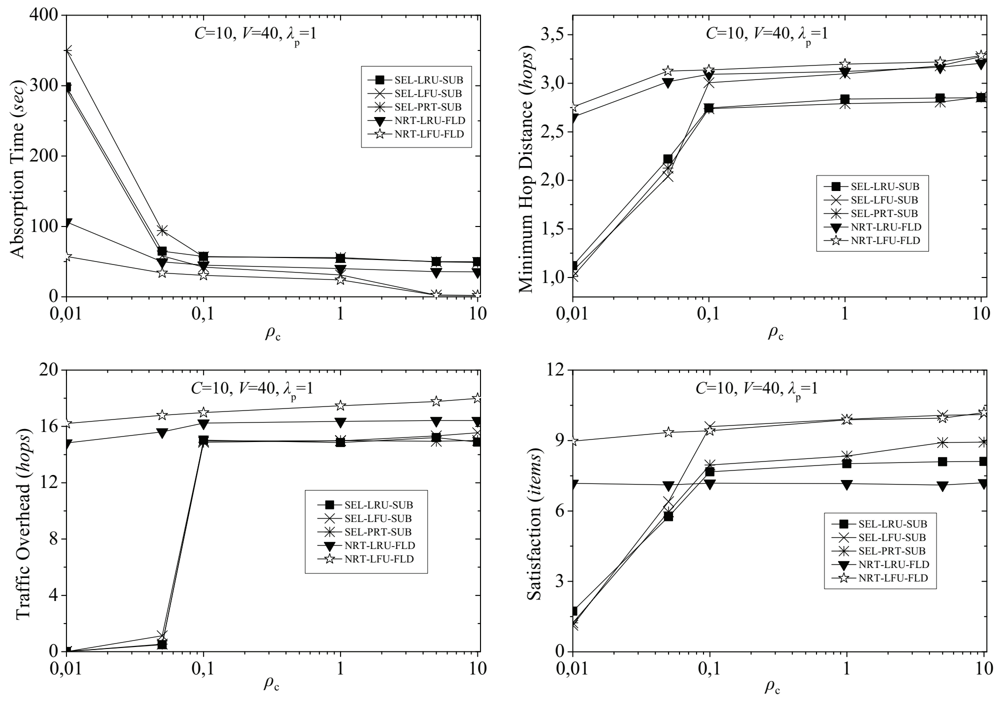

Figure 5.9: (a) Absorption Time, (b) Minimum Hop Distance, (c) Traffic Overhead per Request and (d) Satisfaction vs ρ_c.

5.5.3 Simulations Varying the Subscribers' Intensity per Node

In the following sections we evaluate the proposed opportunistic caching mechanism using a discrete event simulator. We used network topologies from the Internet Topology Zoo dataset [50], which contains real network topologies from all over the world. Since for a given network size more than one topologies may exist in the dataset, we used all of them for averaging purposes and each point of the following figures in the mean value of the different experiments executed for each topology. We are looking at the following interesting metrics:

- The *Mean Absorption Time* of the item is the mean time passed from its publication until it disappears from the network. This metric is indicative of the capability of the network to maintain information items in its memory.

- The *Minimum Hop Distance* is measured for each successful response and corresponds to the minimum number of hops between a responding node and the node where the subscriber making the request is attached to. This metric is indicative of the delay of responses as a function of hops in the network.

- The *Traffic Overhead* is measured for each successfully responded request and is the total number of hops that the duplicate responses travel in the network until the two duplicate dropping mechanisms (proposed in Section 5.2.1) discard them.

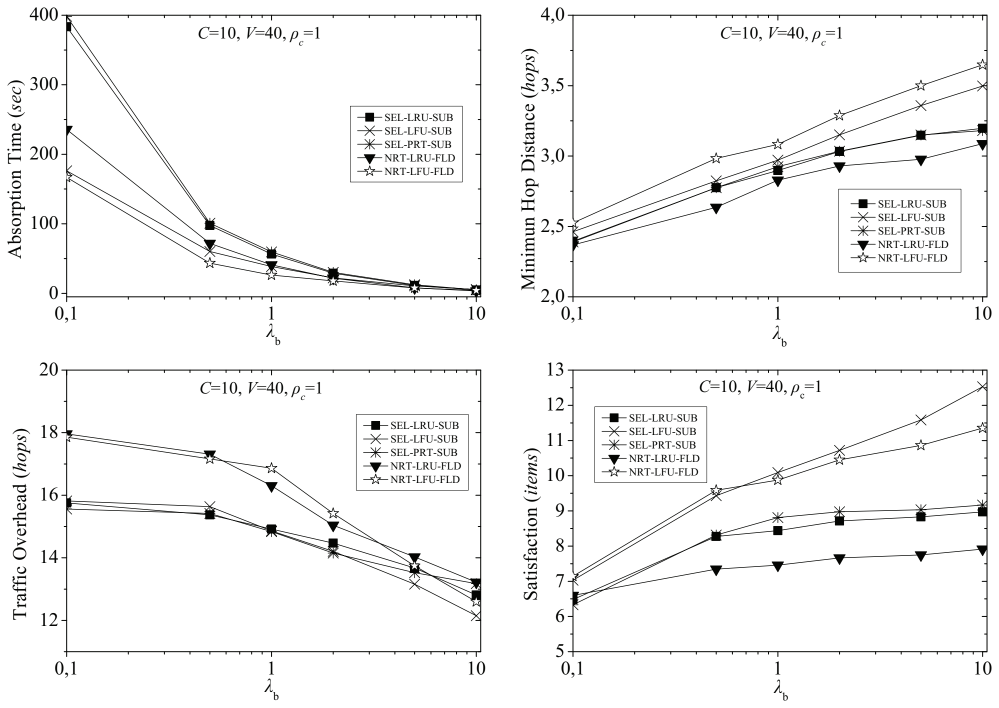

Figure 5.10: (a) Absorption Time, (b) Minimum Hop Distance, (c) Traffic Overhead per Request and (d) Satisfaction vs λ_b.

- The *Satisfaction* is the total number of cached information items (not duplicate) retrieved for each request.

The above metrics are random variables and we estimate their mean by simulating thousands of observations. We set out four experiments one varying the client intensity per node ρ_c, one varying the publication rate λ_b, one varying the number of nodes in the network V and finally, one varying the number of cache slots C per node.

In every other experiment we assume that in the $\rho_c = \lambda_c/\mu_c$ the $\mu_c = 1$ (req/sec) is the same for every item and the λ_c shown at every figure is the request rate per node for the most popular item. The request rate of the other items and consequently the λ_c are given by the Zipf distribution of their popularity. Figure 5.9(a) shows that the Absorption Time decreases fast with ρ_c for every caching mechanism. This is also evident from Eq. (5.5) since an increase in ρ_c increases the rate at which an item is pushed at the end of the cache and eventually is discarded. The Minimum Hop Distance (Figure 5.9(b)) is only affected by the ρ_c when we use the selective caching policy and only when $\rho_c \cdot V < 1$. When $\rho_c \cdot V > 1$ on average there are always subscribers for a given item that is reachable by a request, so when $\rho_c \cdot V < 1$ the requests cannot retrieve distant items (as also depicted in the Figure 5.9(d)). Also the ρ_c does not affect the Traffic Overhead (Figure 5.9(c)). The low overhead when the selective caching is used and $\rho_c \cdot V < 1$ arises also from the fact that requests don't travel far in the network. It is worth mentioning that the two duplicate dropping mechanisms prevent the excessive amount of overhead responses and that is the reason why the en-route caching policy, where

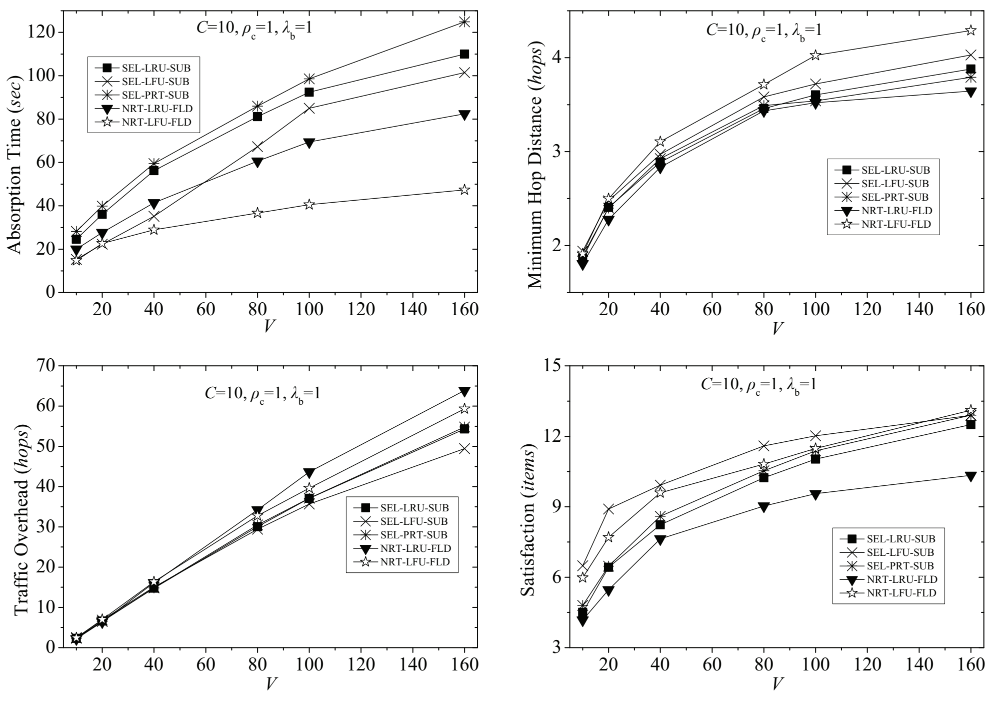

Figure 5.11: (a) Absorption Time, (b) Minimum Hop Distance, (c) Traffic Overhead per Request and (d) Satisfaction vs V (nodes in the network).

multiple duplicates of the same item appear in the network, does not perform very bad. Regarding the number of retrieved items per request (Satisfaction Figure 5.9(d)) it is obvious that in the same way is not affected by ρ_c. From Figure 5.9 is obvious that the proposed selective caching policy especially when combined with the the newly proposed subscription-based request policy and the priority placement policy is on average better than the rest examined caching mechanisms.

5.5.4 Simulations Varying the Publication Rate

Figure 5.10(a) also shows that the Absorption Time decreases fast with λ_b for every caching mechanism since, large values of λ_b means that more and more items have to be cached every time and replace already cached items. This is also evident from Eq. (5.4) and particularly form the rate λ_h, which is the rate at which an item is pushed in the cache when a new publication occurs. The Minimum Hop Distance (Figure 5.10(b)) is not severely affected by the λ_b. Particularly, it slightly increases since the publication of new items allows request packets to find unique (not duplicate) items further in the network. The Traffic Overhead is also decreased significantly (Figure 5.10(c)), since the large number of new published items don't allow the existence of many duplicates of the same items in the network.

Regarding the Satisfaction metric, we observe a linear increase in the number of returned responses for every caching mechanism due to the increase in the number of unique items cached in

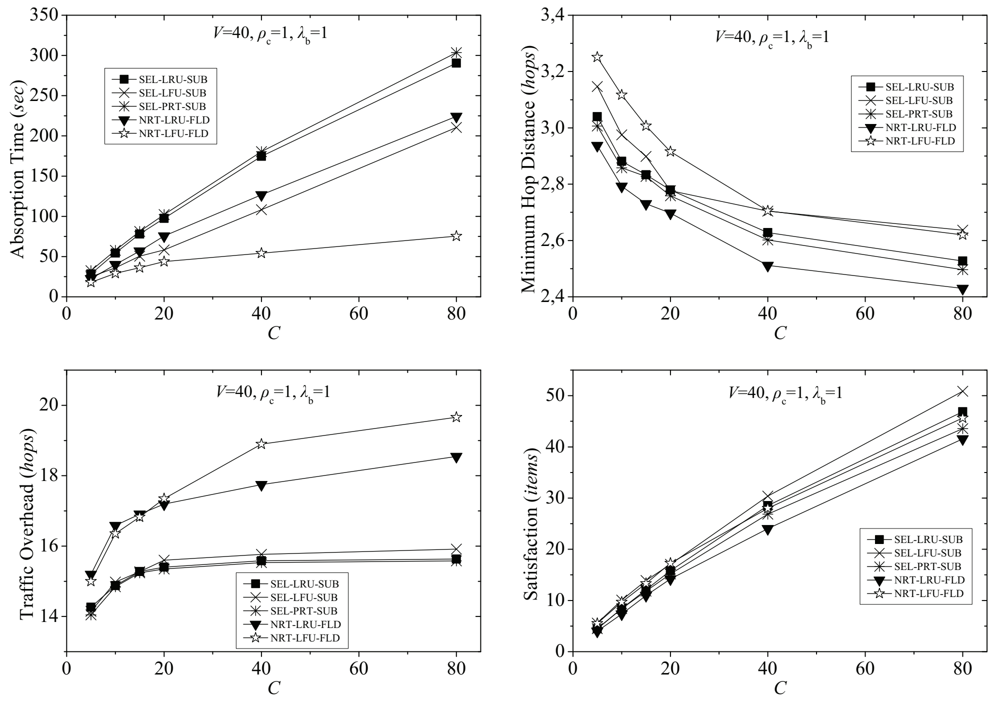

Figure 5.12: (a) Absorption Time, (b) Minimum Hop Distance, (c) Traffic Overhead per Request and (d) Satisfaction vs C (cache slots per node).

the network. From Figure 5.10 is obvious that the proposed caching mechanism SEL-PRT-SUB is on average better than the rest examined caching mechanisms, especially in the cases of the Absorption Time and the Satisfaction metrics. The performance of those metrics is actually the main target of the work presented in this chapter.

5.5.5 Simulations Varying the Number of Nodes in the Network

Figure 5.11(a) shows that the Absorption Time increases almost linearly with V, since more nodes means more caching points leading to more content replicas available in the network and longer survival time of an item in the system. This is also evident from Section 5.5.2 where large V means larger $R_m(\mathbf{s})$ (number of copies of item m in the network) which implies more time for the item to get absorbed. The Minimum Hop Distance (Figure 5.11(b)) is increasing with the increase of the number of nodes in the network, since now items are also fetched from more distant nodes. Traffic Overhead is also increased linearly, but the two duplicate dropping mechanisms prevent the excessive amount of overhead responses, especially when the en-route caching policy is used, retaining the overhead only $5\% - 25\%$ extra overhead responses than the selective caching policy (Figure 5.11(c)).

Regarding the number of retrieved items per request the increase of the number of nodes has not such a significant impact (double responses when the number of nodes is sixteen times larger). More nodes might mean more different cached items, but on the other hand more nodes also increase

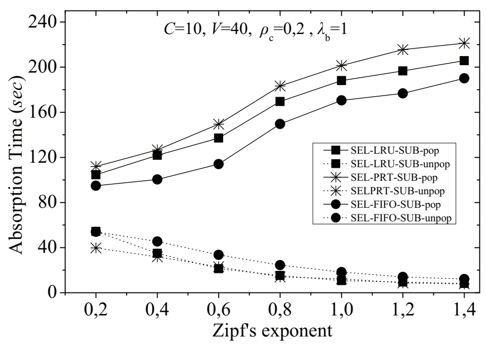

Figure 5.13: Performance evaluation in case of different item popularity.

the replication degree of the cached items ($R_m(\mathbf{s})$) which does allow the retrieval of more unique cached items (Figure 5.11(d)). Figure 5.11 also shows that the proposed caching mechanism SEL-PRT-SUB is on average better than the rest examined mechanisms and maintain the functionality of the used architectural design, being slightly worse regarding Satisfaction from the schemes that use the flooding request policy.

5.5.6 Simulations Varying the Number of Cache Slots per Node

Figure 5.12(a) shows that the Absorption Time increases linearly with C, since more cache slots per node means more caching points and longer stay of an item in the system. This is also evident from Eq. (5.4) where increasing C increases all the transition rates that maintain a given item m "higher" in the cache and further from being dropped. The Minimum Hop Distance (Figure 5.12(b)) is also improved but not significantly by the increase of the number of the caching slots per node (only 20% decrease when we increase C sixteen times) since this metric is mainly affected by the size of the network (number of nodes and topology). Traffic Overhead is also slightly increased by the increase of the caching capability of the nodes (Figure 5.12(c)) since more cache slots allows the caching of new and not duplicate items.

Regarding the Satisfaction metric the increase of the number of cache slots per node has significant impact in the number of retrieved items, since now more unique items can be cached in the network (Figure 5.12(d)). From Figure 5.12 is obvious that the proposed caching mechanism SEL-PRT-SUB is on average better than the rest examined mechanisms maintaining at the same time the basic functionality of the the used architectural design. Particularly, behaves better regarding Absorption Time than any other mechanism, equally regarding Satisfaction to the mechanisms that use en-route caching and flooding request and on average 50% better regarding the Traffic Overhead.

5.5.7 Simulations Varying the Popularity of Items

In Figure 5.13, we compare the performance of the proposed selective caching policy combined with the LRU, the priority replacement policy and the plain FIFO replacement policy when the Zipf expo-

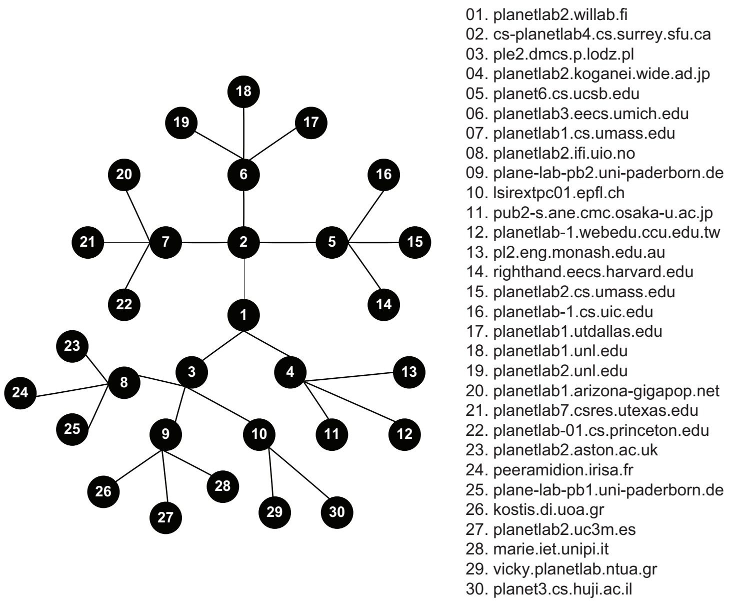
01. planetlab2.willab.fi
02. cs-planetlab4.cs.surrey.sfu.ca
03. ple2.dmcs.p.lodz.pl
04. planetlab2.koganei.wide.ad.jp
05. planet6.cs.ucsb.edu
06. planetlab3.eecs.umich.edu
07. planetlab1.cs.umass.edu
08. planetlab2.ifi.uio.no
09. plane-lab-pb2.uni-paderborn.de
10. lsirextpc01.epfl.ch
11. pub2-s.ane.cmc.osaka-u.ac.jp
12. planetlab-1.webedu.ccu.edu.tw
13. pl2.eng.monash.edu.au
14. righthand.eecs.harvard.edu
15. planetlab2.cs.umass.edu
16. planetlab-1.cs.uic.edu
17. planetlab1.utdallas.edu
18. planetlab1.unl.edu
19. planetlab2.unl.edu
20. planetlab1.arizona-gigapop.net
21. planetlab7.csres.utexas.edu
22. planetlab-01.cs.princeton.edu
23. planetlab2.aston.ac.uk
24. peeramidion.irisa.fr
25. plane-lab-pb1.uni-paderborn.de
26. kostis.di.uoa.gr
27. planetlab2.uc3m.es
28. marie.iet.unipi.it
29. vicky.planetlab.ntua.gr
30. planet3.cs.huji.ac.il

Figure 5.14: The used Planetlab nodes, arranged in one of the emulated topologies.

nent of the item popularity changes. Particularly, we compare the most and least popular items with respect to their mean Absorption Time. First, we observe that the system promotes the differentiation of the items by reducing significantly the Absorption Time of the unpopular item and allocating more cache space to the popular one. Next, by comparing the policies it is possible to quantify the gain from the prioritization inside the cache (regenerations and degradations). We conclude that our mechanism that combines priority replacement and selective caching provides an important tool for differentiating the items based on their popularity. Finally, it is obvious that even the usage of the FIFO policy is good enough to provide prioritization inside the cache revealing the implicit prioritization capabilities of the newly proposed selective caching policy.

5.6 System Design And experimentation

In this section, we describe our prototype implementation of the proposed opportunistic caching mechanism using the REDS overlay CBPS system [17]. REDS is a framework of Java classes to build information-centric applications for large, dynamic networks. REDS provides a modular architecture whose components can be easily changed to adapt to different deployment scenarios. REDS' Java interfaces and classes, define: i) the subscriber API, enabling access to the information-centric services, ii) the node API, enabling access to the components inside the node.

Nodes in REDS are structured in a set of modules grouped in two layers: the Overlay and the Routing layer. The first manages the overlay network that interconnects nodes, while the second is in charge of routing packets. Programmers using REDS may adapt the behavior of the system to their needs, while they are free to implement new versions of the components if those do not fit their needs. We modified the REDS system in order to implement the newly introduced packets and support the Cache and the Pending Request Table. Apart from those changes we also modified the Overlay layer in such a way that the new packets can be transferred. Changes have also been made to the subscriber API and particularly to the functions that generate/manipulate the request and the response packets.

In order to evaluate our system prototype, we have deployed it and performed experiments in PlanetLab [85] using 30 sites (nodes) around the globe. PlanetLab is a global research network that supports the development of new network services such as distributed storage, network mapping, peer-to-peer systems, distributed hash tables, and query processing. Those sites form the network and we emulated topologies from the Zoo dataset. Figure 5.14 illustrate the used Planetlab nodes, as well as an emulated topology. As with the simulations, subscribers are dynamically deployed using a birth and death process emulated by Java applications generating threads for subscribing/requesting to and unsubscribing from items they are interested in at every node of the overlay with specific rates. We set three experiments, one varying the subscribers intensity per node per item ρ_c, one varying the publication rate λ_b and one varying the cache size C of each node. Apart from the Traffic Overhead and the Satisfaction (that are similar to the simulation experiments) we are also looking at the following interesting metrics:

- The *Cache Hit Ratio* is the ratio between the requests found at least one matching item (unique) over the total requests exchanges between the nodes in the network.

- The *Duplicate Dropping Ratio* is the ratio of the dropped responses (using the two duplicate dropping mechanisms presented in Section 5.2.1) over the total responses exchanged between the nodes of the network. This metric shows how efficient are the proposed dropping mechanisms and the overhead generated by each caching mechanism.

5.6.1 PlanetLab Experiments Varying the Subscribers' Intensity per Node

Figures 5.15(b) and (c) show that the Traffic Overhead and the Satisfaction metrics are inline with the corresponding figures from the simulation experiments. The PlanetLab experiments show that the Traffic Overhead and the Satisfaction are not severely affected by the subscribers intensity. Also the caching mechanisms which use the en-route caching policy and the flooding request policy manages to return slightly more items at the cost of higher Traffic Overhead. Of course the duplicate dropping mechanisms keeps that extra overhead only $15\% - 20\%$ more than the overhead using the mechanisms with the selective caching and the subscription based request policy. Regarding the new metrics, Figure 5.15(a) shows that the proposed selective caching policy combined with the subscription based request policy yields almost 20% more cache hits since the en-route counterpart tend to find more duplicate items. This is also obvious from Figure 5.15(d) where the duplicate dropping mechanisms need to drop at least 20% more responses when the flooding request policy is used. Figure 5.15, as previously, shows that the proposed mechanism SEL-PRT-SUB is on average better than the rest examined mechanisms.

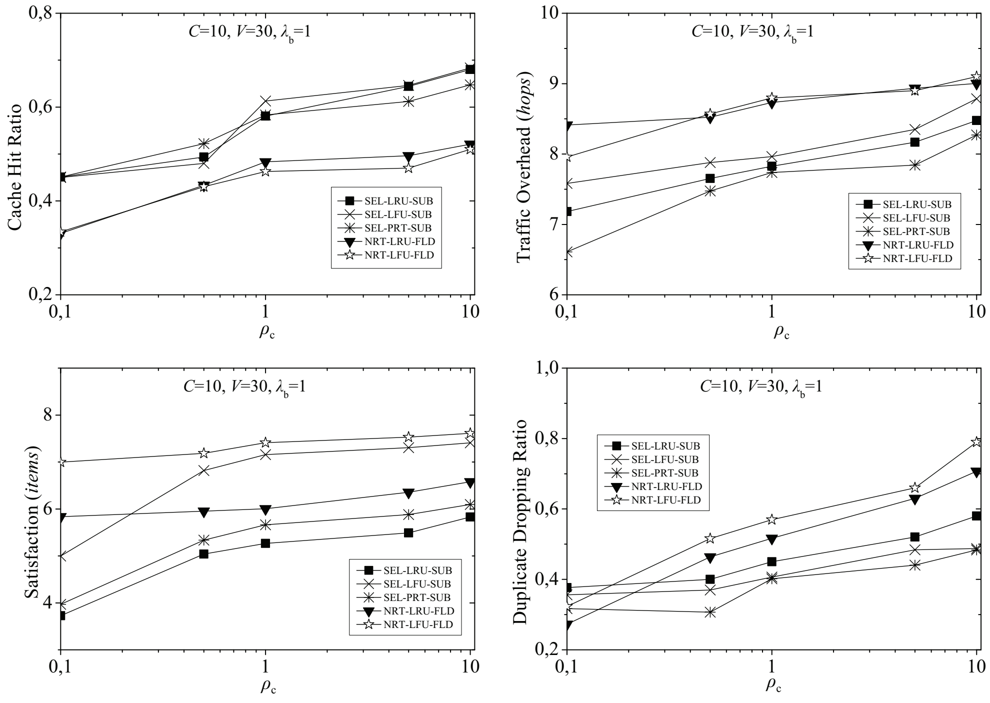

Figure 5.15: (a) Cache Hit Ratio, (b) Traffic Overhead per Request, (c) Satisfaction and (d) Duplicate Dropping Ratio vs ρ_c.

5.6.2 PlanetLab Experiments Varying the Publication Rate

As previously, Figures 5.16 (b) and (c) presenting the Traffic Overhead and the Satisfaction metrics are in conformance with the corresponding figures from the simulations depicting the same metrics. This is another proof that our implementation manages to capture all the characteristics of the examined caching mechanisms. Regarding the Cache Hit Ratio presented in Figure 5.16(a) we observe that the publication of new items tend to remove faster the cached ones, decreasing the possibility of finding matching items on them. Of course when new information items are published very fast the Satisfaction metric might slightly increase but, especially when the en-route caching policy is used, the multiple caching of the same items in the network dictates the duplicate dropping mechanisms to drop almost $60\% - 90\%$ of the generated responses. In other words, the high publication rate especially when the en-route caching is used degrades the placement/replacement policies into the plain FIFO policy and the levels of differentiation that they offered among the items (popular and unpopular) is diminished. Figure 5.16 is another proof that the proposed caching mechanism SEL-PRT-SUB is on average better than the rest examined mechanisms maintaining at the same time the functionality of the used architectural design.

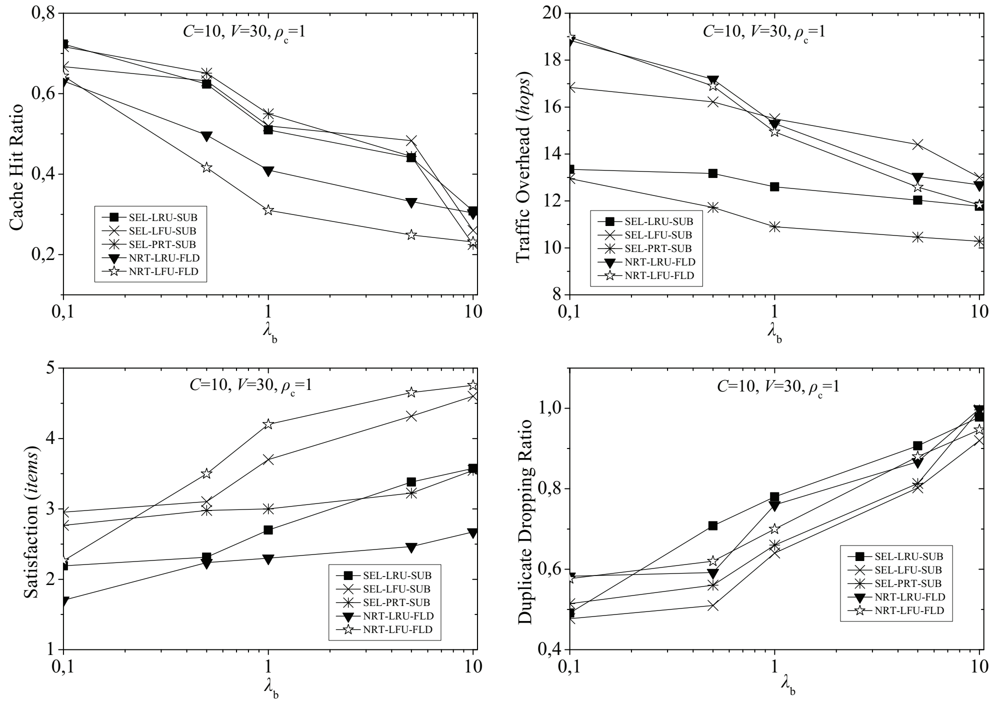

Figure 5.16: (a) Cache Hit Ratio, (b) Traffic Overhead per Request, (c) Satisfaction and (d) Duplicate Dropping Ratio vs λ_b (publication rate).

5.6.3 PlanetLab Experiments Varying the Number of Cache Slots per Node

As in the previous two cases the results from the PlanetLab are in perfect conformance with the corresponding results obtained by the experiments in the discrete event simulator. The Cache Hit Ratio (Figure 5.17(a)) is increasing as we increase the cache capability of each node, since now more different items could be cached at each node of the network. Of course, increasing the cache capacity of each node has as side effect the survival in the network of more duplicate items which increases the Duplicate Dropping Ratio (Figure 5.17(d)).

5.7 Mobility Support Through Opportunistic Caching

The majority of the proposed information-centric approaches are designed not to tolerate any form of topological reconfiguration, therefore they cannot be exploited in those application scenarios where decoupling would be most beneficial. The first information-centric system that supported mobile subscribers was JEDI [86], where subscribers used two functions (*move-out* and *move-in*) to explicitly detach from the network and reconnect to it, possibly through a different node. In [87] authors implement a mobility support service that is independent of the underlying overlay and transparently manages active subscriptions and incoming items when a subscriber detaches from one node until it reattaches at another. They use mobile service proxies which are independent, stationary components

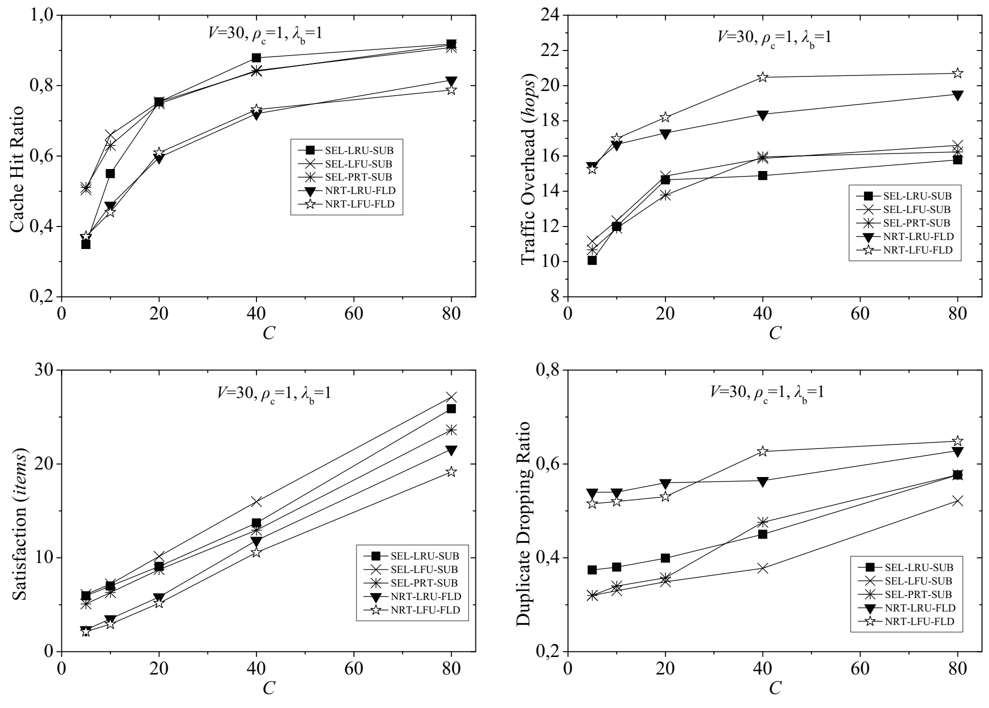

Figure 5.17: (a) Cache Hit Ratio, (b) Traffic Overhead per Request, (c) Satisfaction and (d) Duplicate Dropping Ratio vs *C*.

that run at the edges of the network. In other words they use a second overlay network to take care the mobility of the subscribers. That second overlay is responsible to gather the publications, that match the interests of the mobile subscriber, and deliver them when they reconnect to the network. The proxies of that second overlay should be aware of the topology of the mobile service network, since they should directly contact each other when a subscriber moves among them. Finally in [88] authors present COMAN (COntent-based routing for Mobile Ad-hoc Networks), a protocol to organize the nodes of a MANET in a tree-shaped network able to self repair to tolerate the frequent topological reconfigurations. COMAN was designed to minimize the number of nodes whose routing information are affected by topological changes, but it does not support the retrieval of lost items after the reconfiguration of the network.

In this section we are interested in supporting the mobility of subscribers, where a subscriber is disconnecting from the network and reconnects from a different point later in time. Particularly, we propose a modification to the caching mechanism described in this chapter to enable mobility of the subscribers. Our approach is relative to [87], with the important difference that no extra functionality is required. Particularly, using a portion of each node's cache, we allow caches to manage subscriptions and publications on behalf of the mobile subscribers, both while they are disconnected and during the switch-over phase.

When a subscriber is connected, receives information items directly from the network. Before detaching, the subscriber sends to the node (node 1 in Figure 5.18), that is attached to, a `Request()`

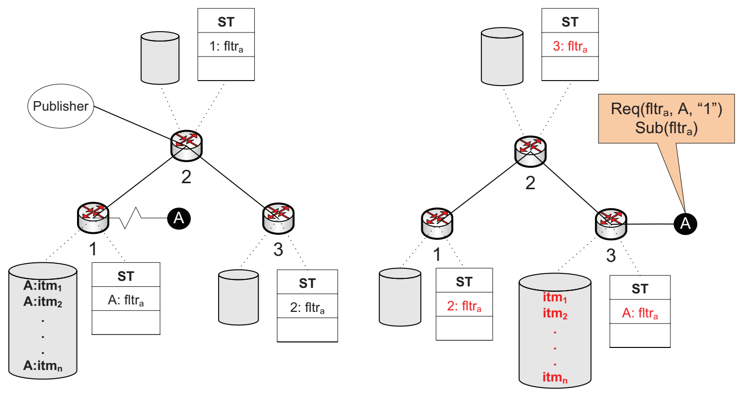

Figure 5.18: Mobility support mechanism.

packet requesting to detach. That request packet is similar to the packet described in the above sections but instead of the requesting filter it contains only the "*id*" of the corresponding subscriber. The node has already in its Subscription Table "ST" the id of the subscriber and its subscription filters so now whenever an item, matching those filters, arrives the node directly caches it (apart from delivering it to the rest of the connected subscribers, if any, with a matching subscription). Until now the procedure is exactly the same with the procedure of the caching mechanism described above. The difference appears in the treatment of those cached item.

In order to make the mobility support robust, we equip the caches with a preemption priority mechanism for those items that are cached for a mobile subscriber. Using such a mechanism, these items are cached in a FIFO manner disregarding the rest of the items which contend only for the remaining cache slots. Thus, an item cached for a mobile subscriber is only dropped from the cache when the cache is full with such items, that arrived later than the given one.

When the mobile subscriber reconnects to the network, from a different node (node 3 in the example), issues a `Request()` packet with the subscription filter (or filters or part of them) that had subscribed before the movement and the "*id*" of the node that was connected (node 1 in the example). Node 1, upon receiving that request, will i) respond with the cached items (items that arrived when the subscriber was in movement, itm_1 - itm_n in the example), ii) unsubscribe the mobile subscriber from its Subscription Table and iii) remove those items that no other subscriber has matching subscriptions to them. This means that those items are treated according to any other item in the cache.

5.7.1 Testbed Demonstration of the Mobility Support Mechanism

We used the same prototype implementation presented in Section 5.6. We used 5 laptops, the 3 were connected via Ethernet switch to set the network as shown in Figure 5.18. Another computer played the role of publisher while the final laptop played the role of a mobile subscriber. In our testbed experiments, the mobile subscriber issues one subscription while a series of publications are published (all publications match that subscription) at a constant rate of λ_b publications per second

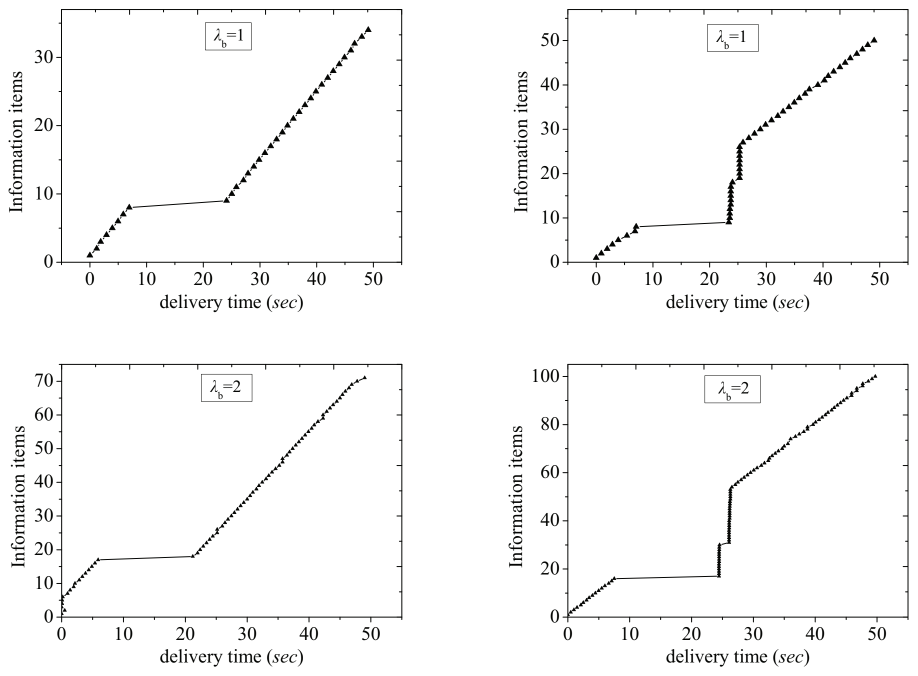

Figure 5.19: No mobility support (left-side), mobility support (right side).

($\lambda_b = 1$ and 2), which lasts for 50 seconds (50 and 100 items accordingly). The mobile subscriber disconnects from one node and connects to another only once, while the mobility interval is fixed and equal to $\Delta t = 15$ sec.

The y axis of Figure 5.19 corresponds to the number of items delivered to the mobile subscriber, while the delivery time is when the item is received by him (set time to zero when the first item is delivered). To the left, the case where no mobility is supported is showcased, while to the right, the effect of our mobility support mechanism is demonstrated. Every point in the figures corresponds to an item received by the subscriber either through the publish or the request process. The part of the figures where there is no item delivery represents the time interval that the subscriber is disconnected from the network, while the vertical part of delivered items (right-side figures) after the reconnection of the subscriber represents the responses delivered to the subscriber to his request sent to the node that was attached before the movement. It is obvious that all the published items finally are delivered to the client.

5.8 Chapter Conclusions

In this chapter, we put forward a new mechanism for in-network opportunistic caching in information-centric peer-to-peer networks. Particularly, we enhanced the CBPS architectural design with a request/response scheme so that subscribers can retrieve previously published items. The proposed caching mechanism maintains the loose-coupled and asynchronous communication of the CBPS model since there are not assumed predefined caching points and the requests for cached items are handled transparently by the network. We also proposed a stochastic model that captures the dynamics of the newly proposed caching mechanism. Evaluation through simulations and system prototype experimentation in PlanetLab shows that the proposed caching mechanism outperforms traditional caching mechanisms retaining at the same time the traffic overhead in low levels. Moreover, two duplicate dropping mechanisms have been proposed to diminish the amount of duplicate responses in the network. Finally, we presented a modification to the proposed caching mechanism to support mobility of the subscribers.

Chapter 6

Cache Aware Routing

The research issues addressed by most of the proposed ICN architectures are related to persistent/unique naming, efficient content distribution and discovery through name-based addresses, in-network caching and security. Given this emergence of ICN oriented solutions, the relevant management needs in terms of performance have not been extensively studied with most efforts focusing on the performance of the in-network caching schemes. Moreover, little attention has been given on designing efficient routing mechanisms suitable for ICN, since most of the approaches assume either traditional shortest path or inefficient flooding schemes without taking into consideration the caching capability of the used path and the possibility of using an alternative routing scheme. In this chapter, we propose an intra-domain cache aware routing scheme that computes the paths with the minimum transportation cost based on the information item demands and the caching capabilities of the network, we derive analytically the communication and computational complexity of the proposed approach and we evaluate its performance through simulations.

6.1 Introduction

Given this emergence of ICN oriented solutions, the relevant management needs in terms of performance have not been adequately addressed, yet they are absolutely essential for relevant network operation and crucial for the ICN approaches to succeed. Performance management and traffic engineering approaches are also required to control routing, to configure the logic for replacement policies in cache enabled nodes and to control decisions where to cache, for instance. While quite a lot of studies have been performed related to caching, routing functionality is completely missing from the current ICN design with only simple flooding or OSPF-like shortest path mechanisms having been proposed. This choice has been deliberately left open in order to allow routing solutions ranging from schemes potentially based on known protocols to innovative solutions best suited to the specific communication model of ICN.

In the area of routing in ICN authors in [89] present what inter-domain routing policies could look like in an NDN Internet (policies are realized by tuning a variety of parameters or *knobs*, available in the Internet BGP routing protocol). They describe the knobs available to network operators and their possible settings and they explore new economic incentives that may be present in an NDN Internet and consider what types of routing policies they may develop. In [90] authors propose a scalable routing and name resolution framework, called *Scalable Multi-level Virtual Distributed Hash Table* (SMVDHT). SMVDHT uses a combination of name aggregation and multilevel virtual DHTs to achieve scalability. SMVDHT also exploits underlying intra- and inter-domain IP routing protocols to build multi-level virtual DHTs for name resolution, which is more efficient than conventional hierarchical DHT schemes and simplifies network management. Finally, authors in [91] propose the *Potential Based Routing* (PBR) in ICN to achieve several design goals such as availability, adaptability, diversity, and robustness. The proposed PBR is more close to the service discovery mechanism [92] which has been applied to mobile ad-hoc networks.

Moreover, in [93] authors study the viability of joint use of load-balancing, multipath routing and caching in the CCN architecture. Particularly, they classify the information items in three popularity groups based on a popularity estimation mechanism and they propose a *popularity aware load balancing scheme* (PALB) in order to maximize the effectiveness of caching while balancing link load. In more details, all the interests for a high/medium popular item are transmitted over a unique tree towards the origin server in order to maximize the probability that this item is cached in the network, while the interests for the least popular items use random paths in order to eliminate the probability to be cached. The selection of the outgoing interface over which an interest for a popular item is to be sent is decided according to a *modulo hash first* (MHF) load-balancing scheme. Additionally, in [94] authors propose a simple content caching, location and routing system that adopts an implicit, transparent and best-effort approach towards in-network caching. Each cache equipped node sets aside some of its cache space for the purpose of storing routing history, or *breadcrumbs* (BC), of previously seen files. The proposed *Best Effort Content Search* (BECONS) query routing policy forwards a request/query for an information item either upstream towards the origin server or downstream the breadcrumb trail in an attempt to locate the item faster and closer to the requesting node. Though best effort the proposed policy outperform classic policies (e.g. route only to the source) or policies with explicit coordination between caches.

Finally, in [95] authors investigate whether hash-routing [96] is a viable and efficient caching approach when applied outside enterprize networks, but within the boundaries of a domain. Every edge domain router implement a hash function that determines both the placement of the content across the in-network caches of a domain and how content requests are resolved to the corresponding cache nodes. Each information item can be cached in a domain at most once, thus preventing redundant replication of cached content and resulting in a more efficient utilization of cache space. This approach also allows edge routers to forward content requests to the designated cache directly, without performing any lookup. In particular, when the edge routers of a domain receive a content request, they calculate the hash of the content identifier and redirect it to the responsible cache. In the case of a cache hit, the content is returned to the client, otherwise, the request is forwarded towards the original server. Similarly, incoming contents are forwarded according to the hash of their identifier. The five proposed hash-routing schemes manage to reduce significantly the inter-domain traffic by increasing the cache hit ratio within the domain with minimal impact on the traffic dynamics of intra-domain links. A similar approach to increase content availability through hash-routing, regardless of the increase in the overall network traffic, has also been presented in [97], where authors also present an extension to inter-domain scenarios. All the related work presented above aims at improving content availability inside a domain, i.e. maximising cache hit ratios even by following longer routes to an available cache, thus increasing the overall network utilisation, while the optimisation objective of our routing scheme is to minimize the overall network traffic taking into account the caching capabilities of the nodes.

In this chapter, we propose a new cache aware intra-domain routing scheme that dynamically computes the routes followed by each request for each item and from each node in the network. Particularly, we present a dynamic programming (DP) approach for the computation of the minimum transportation cost paths based on the observed item request patterns, such as their popularity and locality and the used caching scheme, in order to minimize the overall transportation cost imposed by the user requests. Moreover, we propose an iterative algorithm for the computation of the minimum cost transportation paths for those scenarios where the routing decisions interact with the caching decisions. Finally, we shortly present a resource management system architecture for the cache aware routing in ICN, where resource managers make route decisions, and we validate the proposed scheme through simulations providing also insight on its ability to adapt to the ever-changing ICN environment caused by the volatility of the user requests.

The rest of the chapter is organized as follows. In Section 6.2, we present the functionality of the resource management architecture that will take care the cache aware routing decisions in ICN based on the selected caching scheme and the volatility of the user requests. In Section 6.3, we formulate the cache aware routing problem and present the DP approach for the computation of the minimum transportation cost paths given the cache hit probability of each item at each node of the network. Moreover, in Section 6.4 we present the iterative algorithm for those scenarios where the cache hit probability of each item depends on the routing decisions. Finally, in Section 6.5 we evaluate the performance of the proposed routing scheme and we compare its outcome with the traditional shortest path routing scheme, while in Section 6.6 we conclude the chapter.

6.2 Resource Management System Architecture

Despite the fact that in-network caching has emerged as one of the most important research fields in the context of ICN, most of the research attempts assumed the use of the shortest (in terms of hops) path as the delivery path and attempt to optimize the network performance through the exploration of various caching schemes in the nodes, without exploring at the same time the caching capabilities of alternative possibly longer paths. Consequently, efficient management of such networks entails managing both the routing and the caching scheme used at each node with objectives such as minimizing the content access latency from clients, maximizing the traffic volume served by caches and thus minimizing the bandwidth cost and congestion of the server.

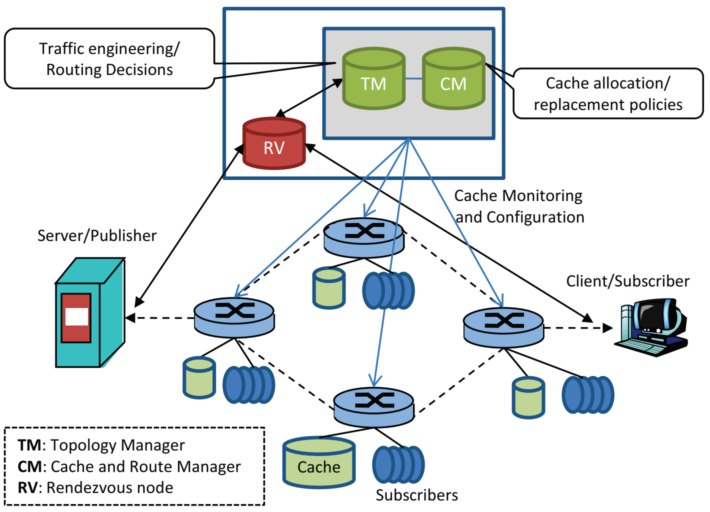

Figure 6.1: ICN resource management in a PURSUIT-like network architecture.

In this section, we present an ICN resource management system architecture for controlling the routing processes as well as the cache resources of the network. In Figure 6.1, we depict how such a system can be deployed on top of a PURSUIT-like network architecture following a centralized approach, while a distributed approach applied on a NDN-based network is shown in Figure 6.2.

Although, the PURSUIT architecture originally followed a PUSH communication model, in order to support opportunistic in-network caching a PULL model has also been described in [70]. In this ICN implementation, the entities that decide on the routing and caching scheme adopted in the managed network are called Topology Manager (TM) and Cache and Route Manager (CM) and reside either in a node of the network or in a management server and can be co-located with the Rendezvous Node. The TM/CM may either take long term traffic engineering decisions based on predicted information item demands or dynamically control the routing and cache resources based on real-time network information by monitoring the status of the network e.g. cache hit probabilities, link utilization, etc. The TM/CM can extract the demand patterns by monitoring the RV node, which gathers the subscriptions of every user in the network in order to bind the publishers to the subscribers, and passes the successful bindings to the TM to compute the paths from the subscribers to the publisher(s), based on its optimization objectives.

In NDN [13] when a user wishes to receive data, he/she issues an Interest packet that contains the data name. The network propagates the Interests to the nearest data source (anycast) and then the requested item is delivered back to the user in the form of a Data packet. Each node uses the Pending Interest Table (PIT) to keep track of the forwarded Interests. In Figure 6.2, a distributed approach is depicted, where a Resource Manager (RM) is installed in every node of the network. The RMs are responsible for computing the routes to the server(s) of the information items and enforce cache allocation and replacement policies in the co-located node. Their decisions may be based on local information i.e. by monitoring the PIT table to estimate the local demands, cache hit probabilities,

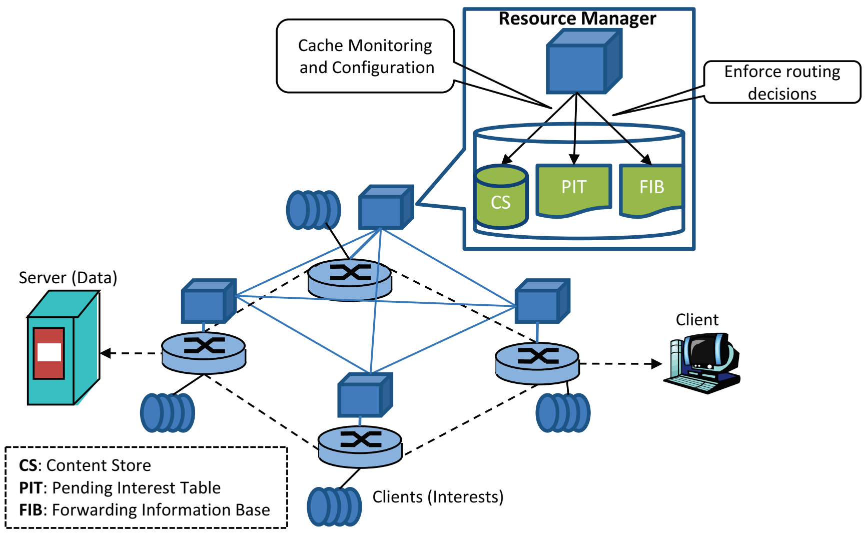
Resource Manager
Cache Monitoring and Configuration
Enforce routing decisions
CS
PIT
FIB
Server (Data)
Client
Clients (Interests)
CS: Content Store
PIT: Pending Interest Table
FIB: Forwarding Information Base

Figure 6.2: ICN resource management in a NDN-based network architecture.

replacements or relevant information received by adjacent nodes for acquiring a network-wide view and take co-operative decisions with the other managers. In NDN nodes, the routing decisions are enforced by configuring the Forwarding Information Base (FIB) table of the NDN node so that Interests follow the paths that the manager has computed. The RM also configures the Content Store (CS) management interface with the appropriate allocation, partitioning and replacement directives.

In the above analysis, we described the functionality of the required additional control components and their interactions with the network components of the two ICN architectures, the CM in PURSUIT and the RM in NDN, that are able to extract from the network all the necessary monitoring information regarding the items request pattern and enforce new cache allocation and replacement policies as well as alternative routing schemes. In the next section, we describe a routing algorithm appropriate for ICNs that takes into account cache related information and realizes the logic of the Topology and Resource Managers as described in the above system deployments. The routing decisions aim at minimizing an overall network-wide utility function i.e. the total transportation cost for the delivery of an item from the hosting server to a local client. The proposed routing algorithm is applicable in both (centralized and distributed) ICN architectures, but we present in detail the distributed case, which also requires coordination and exchange of relevant information between the managers.

6.3 Problem Formulation

We consider a network of arbitrary topology, represented by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. $\mathcal{V}$ denotes the set of nodes and $\mathcal{E}$ the communication links interconnecting them. In this chapter we will also use the calligraphic letters to denote sets and the corresponding capitals for cardinality; for example $|\mathcal{V}| = V$. We also denote with $\mathcal{M}$ the set of the M information items available at the network. For simplicity we assume that each item is of unit size.

Requests for content access are generated by the users of the network, with each user being directly connected to a node. Each information items m is stored permanently at a node of the network (server s^m, $s \in \mathcal{V}$) and every request for that item from every other node is headed towards that node.

Every node $v \in \mathcal{V}$ has a cache of size C_v slots (each slot can hold an item) and requests are generated by clients attached to the node with rate $r_v = \{r_v^1, \ldots, r_v^M\}$, where r_v^m denotes the aggregate incoming request rate (in requests per second) at node v for information item m. Each node opportunistically caches passing by items following a given caching scheme and serves requests for items that are cached. Each item $m \in \mathcal{M}$ is cached with probability p_v^m (cache hit probability) at each node $v \in \mathcal{V}$, independently from node to node.

Assume that node v_1 needs to access the information item m and chooses the path $p = (v_1, v_2, \ldots, v_n, v_{n+1})$ (v_{n+1} is the server of item m, $p_{v_{n+1}}^m = p_{s^m} = 1$) from v_1 to v_{n+1}. We will use the notation $v_1 \rightsquigarrow_p v_{n+1}$ to represent a path from node v_1 to node v_{n+1} (generally every path will be denoted by $\rightsquigarrow_p$).

The average number of hops to find item m through path p is:

$$\begin{aligned} H^m(v_1 \rightsquigarrow_p \ v_{n+1}) = 0 \cdot p_{v_1}^m + 1 \cdot p_{v_2}^m \cdot (1 - p_{v_1}^m) + 2 \cdot p_{v_3}^m \cdot (1 - p_{v_2}^m) \cdot (1 - p_{v_1}^m) + \\ + \cdots + n \cdot p_{v_{n+1}}^m \cdot (1 - p_{v_n}^m) \cdot \ldots \cdot (1 - p_{v_2}^m) \cdot (1 - p_{v_1}^m) \end{aligned} \tag{6.1}$$

where n is the number of hops from node v_1 to node v_{n+1} ($n+1$ is the number of nodes along the path).

For the i-th node in the path p, we define:

$$w_{\rightsquigarrow_p}^m(v_i) = \begin{cases} 0 & i = 1, \\ \prod_{j=1}^{i-1} (1 - p_{v_j}^m) & 2 \leq i \leq n+1. \end{cases} \tag{6.2}$$

We call $w_{\rightsquigarrow_p}^m(v_i)$ the *multiplicative weight* of node v_i along path p for the information item m and $w_{\rightsquigarrow_p}^m = \prod_{j=1}^{n} (1 - p_{v_j}^m)$ the *multiplicative weight* along path p. The multiplicative weight of the whole path denotes also the probability of a request to reach the server of the requested item. Note that the multiplicative weight of a node v_i depends only on the cache hit probabilities of the items at the nodes in the path up to that node and not on the cache hit probability of the node itself. Eq. 6.1 is transformed as:

$$\begin{aligned} H^m(v_1 \rightsquigarrow_p \ v_{n+1}) = H_{\rightsquigarrow_p}^m(v_{n+1}) = 0 \cdot p_{v_1}^m \cdot w_{\rightsquigarrow_p}^m(v_1) + 1 \cdot p_{v_2}^m \cdot w_{\rightsquigarrow_p}^m(v_2) + \\ 2 \cdot p_{v_3}^m \cdot w_{\rightsquigarrow_p}^m(v_3) + \cdots + n \cdot p_{v_{n+1}}^m \cdot w_{\rightsquigarrow_p}^m(v_{n+1}) \end{aligned} \tag{6.3}$$

From the above equation is obvious that the average number of hops that a request will travel along path p until item m is found is:

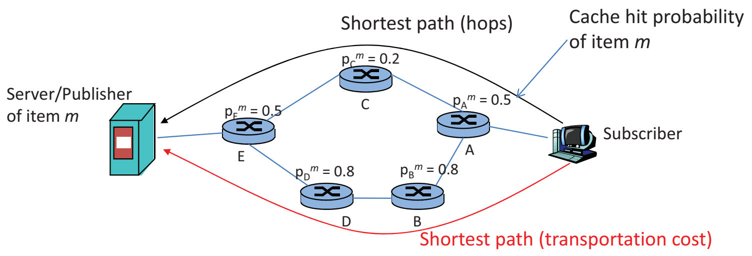

Figure 6.3: A motivation example where the longest path (bottom) produces less traffic in the network than the shortest path (top).

$$H^m_{\rightsquigarrow_p}(s^m) = H^m_{\rightsquigarrow_p} = \sum_{j=1}^{i} (j-1) \cdot p^m_{v_j} \cdot w^m_{\rightsquigarrow_p}(v_j) \quad 1 \le i \le n+1 \tag{6.4}$$

We call $H^m_{\rightsquigarrow_p}$ the *transportation cost* of the unit size item m along path p.

We define the minimum transportation cost of an item $m \in \mathcal{M}$ from node v, where the request was generated, to node s^m (the server node for the particular item) as follows:

$$\delta(v, s^m) = \min\{H^m(v \rightsquigarrow_p s^m)\} = \min\{H^m_{\rightsquigarrow_p}\} \tag{6.5}$$

We assume that the graph $\mathcal{G}$ is connected and there is always a path from node $v \in \mathcal{V}$ to node s^m, $m \in \mathcal{M}$. The statement $\min\{H^m_{\rightsquigarrow_p}\}$ means that a path p with the minimum transportation cost is selected among every path from node v to node s^m. A minimum transportation cost path from v to s^m is defined as a path p with $H^m_{\rightsquigarrow_p} = \delta(v, s^m)$. The cache aware routing problem is then defined as a problem to find the path with the minimum transportation cost for every unit size item $m \in \mathcal{M}$ from each node $v \in \mathcal{V} \backslash \{s^m\}$ to the hosting server s^m.

Consider the motivating example of Figure 6.3 where the request from the subscriber for item m towards the server of that particular item will travel on average 2.1 hops following the shortest (in terms of hops) path, whereas following a potential longer path will actually travel on average 1.63 hops based on the given cache hit probabilities of item m. This means that the longer red path is the path with the minimum transportation cost for the given scenario.

6.3.1 Dynamic Programming approach

In the traditional shortest path problem the weight of each link is static and known a priori. In the above setting the transportation cost between two neighboring nodes depends on their history on the path. From the Eq. (6.2)-(6.4) the transportation cost of item m from node v_i to node v_{i+1} ($(v_i, v_{i+1}) \in \mathcal{E}$) is $i \cdot \left(\prod_{j=1}^{i}(1-p^m_{v_j})\right) \cdot p^m_{v_{i+1}}$. The multiplicative term $i \cdot \prod_{j=1}^{i}(1-p^m_{v_j})$ is the dependence of the transportation cost between the two nodes to the history of the path.

From the formulation of the problem described above is obvious that:

$$H^{m}_{\rightsquigarrow_p}(v_{i+1}) = H^{m}_{\rightsquigarrow_p}(v_i) + i \cdot \left(\prod_{j=1}^{i} (1 - p^{m}_{v_j}) \right) \cdot p^{m}_{v_{i+1}} \tag{6.6}$$

and for the minimum transportation cost of an item $m \in \mathcal{M}$ from node v_1, where the request was generated, to node v_{i+1} (the server node for the particular item) we have:

$$\begin{aligned}
\delta(v_1, v_{i+1}) &= \min \left\{ H^{m}_{\rightsquigarrow_p}(v_i) + i \cdot \left(\prod_{j=1}^{i} (1 - p^{m}_{v_j}) \right) \cdot p^{m}_{v_{i+1}} \right\} \\
&= \delta(v_1, v_i) + \min \left\{ i \cdot \left(\prod_{j=1}^{i} (1 - p^{m}_{v_j}) \right) \cdot p^{m}_{v_{i+1}} \right\} \\
&= \delta(v_1, v_i) + i \cdot \min \left\{ \left(\prod_{j=1}^{i} (1 - p^{m}_{v_j}) \right) \right\} \cdot p_{v^{m}_{i+1}} \\
&= \delta(v_1, v_i) + i \cdot \min \left\{ w^{m}_{\rightsquigarrow_p}(v_{i+1}) \right\} \cdot p_{v^{m}_{i+1}} \qquad (6.7)
\end{aligned}$$

From Eq. (6.2) and (6.7) is obvious that the minimum transportation cost path is the path with the minimum multiplicative weight. In order to minimize the multiplicative weight of a path in Eq. (6.2), we select as the next hop the node with the maximum cache hit probability, since this will minimizes the probability of a request to further be forwarded in the network (an item is more likely to be found at a intermediate node).

From the Dynamic Programming (DP) theory a problem should fulfill two key attributes so that the DP be applicable; optimal substructure and overlapping subproblems. Optimal substructure means that the solution to a given optimization problem can be obtained by the combination of optimal solutions to its subproblems, while overlapping subproblems means that the space of subproblems must be small, that is, any recursive algorithm solving the problem should solve the same subproblems over and over, rather than generating new subproblems. Eq. 6.7 fulfills both attributes and its recursive application for each item $m \in \mathcal{M}$ and for each node $v \in \mathcal{V} \backslash \{s^m\}$ provides the paths with the minimum transportation cost for every item m from every node v, where requests for that particular item were made, to the server s^m of that item. At each recursion the minimum transportation cost towards a neighbor closer to the server is selected.

6.4 Iterative Routing Algorithm

In Section 6.3 for the computation of the minimum transportation cost paths we assumed that each item $m \in \mathcal{M}$ is cached with probability p^m_v at each node $v \in \mathcal{V}$, independently from node to node. The cache hit probability of each item at a given node depends on the caching scheme that is used. The caching scheme consists of two phases; in the first phase it is decided which passing by items to cache (*caching algorithm*) and in the second phase it is decided which cached item should be removed from the cache in case of an overflow (*replacement algorithm*). In [98] authors consider the well known

Least Recently Used (LRU) replacement algorithm combined with local caching, where only the node that hosts the requesting subscriber caches an item, and derive a closed-form formula that can be used for obtaining approximate cache hit probabilities in constant time.

In the approach presented in [98] the cache hit probability of each item at each node is independent of the routing scheme and depends only on the local demand for each item. In this section we present an iterative algorithm for the computation of the minimum transportation cost paths for those caching schemes where the cache hit probability of each item at each node not only depends on the local demand for that item, but also depends on the selected routing scheme (e.g. cache-everything-everywhere caching algorithm [5]). Since there are is no closed-form formula for the computation of the cache hit probabilities of each item in the network we assume that this information is extracted by the CM in the PURSUIT-like architecture (centralized) or the RM in the NDN-like architecture (distributed) after observing the network for a time period T, using the approach described in Section 6.2. Next we present the steps of the proposed iterative algorithm for the distributed scenario.

Step 1: Set $itr = 1$. Observe the network operating for a time period T. At the end of this period each manager at node v computes for each item m its cache hit probability based on the observed incoming requests, and informs the rest managers by sending a report message. For the given cache hit probabilities each manager (installed at node v) execute the DP algorithm, presented in Section 6.3.1, to obtain the minimum transportation cost paths for every item m to the server of item m. Store those paths at $P_v(itr) = P_v(1)$.

Step 2: Set $itr = itr + 1$. Observe the network operating for a time period T (the routing paths are those computed in the previous iteration) and at the end of that period each manager updates the cache hit probabilities of each item m and informs the rest managers.

Step 3: For the given cache hit probabilities each manager executes the DP algorithm to obtain the new minimum transportation cost paths for every item m to the server of that item. Store those paths at $P_v(itr)$.

Step 4: If at each manager the current paths are the same to the paths computed in the previous iteration ($P_v(itr+1) \equiv P_v(itr), \forall v \in \mathcal{V}$) terminate the iterative algorithm, otherwise repeat steps 2-4 until no new paths are computed (no further reduction in the overall total transportation cost is possible).

In a centralized scenario a centrally located component (e.g. TM/CM in PURSUIT) executes the DP algorithm for each one of the nodes without sending report messages to them. Assuming that the user request patterns does not change over the periods that we observe the network the iterative algorithm finally converges to a point where the cache hit probabilities of each item at each node does not change as well. This means that the DP algorithm computes the same paths over the different iterations. In the performance evaluation section we show that the iterative algorithm converges very fast, after a small number of iterations, regardless of the size of the time T that we observe the network. Of course, the larger this time the more accurate is the computation of the cache hit probabilities of each item. The iterative algorithm at convergence computes the paths with the minimum transportation cost, given a caching scheme. Of course, this is not the optimal solution, since such an optimal solution requires also the existence of an optimal caching scheme, which is by itself an NP-hard problem.

6.4.1 Complexity Analysis

In this section we present the communication and computational complexity of the iterative and the DP algorithm presented above. This will give significant insight regarding the incurred computational burden for each distributed manager and the communications requirements of the management architecture. Each manager needs to have a network-wide knowledge of the cache hit probabilities of the items in the nodes of the network. This requires each manager to forward its local vector with the observed cache hit probabilities to all the other managers using a report message. One of the inherent characteristics of the ICN architecture is the multicast nature of the information dissemination. When a manager wishes to disseminate a management message to the rest of the managers in the network this is done through a single transmission over the minimum spanning tree of the network topology (a tree that connects all the nodes/managers). Such a tree has $V-1$ links, where V is the number of nodes in the network, hence a message from a manager to every other manager produces a communication overhead of $V-1$ messages. For this purpose a report message with the cache hit probabilities of size M needs to be forwarded to any other manager, leading thus to a total of $V\cdot(V-1)$ management messages. As a result, the communication complexity of the DP algorithm is $O(V^2\cdot M)$ (assuming that M different report messages have to be sent; each message contains the relative information of only one information item). This is also the per iteration communication overhead of the above iterative algorithm.

For the calculation of the computational complexity of the DP algorithm we firstly define the calculation of the transportation cost from node v_i to node v_{i+1} (the second term in the summation of Eq. 6.7; two simple multiplications) as the basic operation. The DP algorithm executed for a given item m and for every node in the network is similar to any other DP approach for the computation of the shortest path (e.g. single destination shortest path problem) and it requires $O(V\cdot E)$ basic operations, where E is the number of links in the network. Since the DP algorithm should be executed for each item $m\in\mathcal{M}$ the total computational complexity of the DP algorithm is $O(V\cdot E\cdot M)$ basic operations ($O(E\cdot M)$ for each manager). This is also the per iteration computation overhead of the above iterative algorithm.

In order to use an ICN architecture for the dissemination of the management report messages each manager should act both as a subscriber and as a publisher. Particularly, each manager should subscribe to the relative management information of every other manager (e.g. a given scope in the PURSUIT architecture) and should publish its own management information (e.g. under the same scope in PURSUIT) over the ICN, so that it could reach the rest of the managers in the network. In other words, the relative management information that is used by the proposed DP algorithm is treated as another information item by the ICN.

6.4.2 Discussion

In this section, we discuss in detail the fundamental assumptions made throughout the chapter, as well as the applicability of the proposed routing scheme. Each manager requires the r_v vector with the local request pattern in order to execute the DP algorithm. The r_v vector is an estimation of the actual request pattern based on observed, historical data (within a given time window) and this estimation is used as a forecast for the future behavior of the clients attached at each node. The optimal way to perform this estimation is out of the scope of this work, but in an ICN implementation this information could be extracted by the managers monitoring the components described in Section 6.2. In [93] authors propose a method to estimate the number of requests for each item using an exponential moving average function in each measurement window. The proposed iterative algorithm requires the r_v vector of each node to be stable in order to converge. Obviously, if during the iterations the request patterns significantly change, the proposed algorithm is not able to converge. In the evaluation section that follows, we always assume that the request patterns are not changing during the iterations, but we also include an experiment to show how the proposed routing scheme performs when the request patterns change between the iterations.

The proposed cache aware routing scheme is applicable in both centralized and distributed ICN architectures. Of course, in a distributed environment managers should exchange information regarding the topology formation and the cache hit rates for each item at each node. Almost every utility maximization algorithm that has been proposed in the literature, operates in such a distributed iterative way, where nodes exchange information until the algorithm converges. For example both classes of routing protocols used in packet switching networks, (link-state routing protocol and distance-vector routing protocol) operate in a similar way. Of course, our algorithm as already mentioned might not be applicable in Internet scale (also the link-state routing protocol and distance-vector routing protocol are not applicable in Internet-scales). Particularly, from the above complexity analysis we observe that the complexity of the proposed cache aware routing scheme, and as a consequence its future applicability, depends on the number of information items M and the number of the nodes V in the network (consequently the number of links in the network E). In this work, we assume that the proposed cache aware routing scheme is not deployed at full Internet scale but in a domain scale (like all the current ICN implementations), where the number of items and nodes are significantly smaller and hence our scheme is applicable. Moreover, the communication and computational complexity could be further reduced using the appropriate aggregation schemes regarding the naming of items, i.e. scopes in PURSUIT and hierarchical names in NDN.

Finally, since we assume that during the iterations of the proposed iterative algorithm the request patterns of the clients are stable, a centralized off-line approach for the computation of the routing scheme could be adopted in every ICN architecture (centralized or distributed). Particularly, a component similar to the TM/CM in PURSUIT could be deployed in a centralized server e.g. in a NMS (*Network Management System*), where it monitors the items request pattern from all the nodes of the network and enforcing back its decisions. This component, having also the information regarding the network topology and given the utility that needs to be maximized/minimized, can emulate the behavior of the network between the iterations of the iterative algorithm and configure the network nodes with the final routes to be followed by the requests i.e. after the algorithm converges. Of course, such a centralized deployment minimizes the amount of the communication overhead but looses all the benefits of the distributed approach (immunity to node failures and sharing of the computational overhead).

6.5 Performance Evaluation

In this section, we evaluate through simulations (using a discrete event simulator) the performance of the proposed cache aware routing scheme (*CAWR*) and we compare it against the traditional shortest path routing scheme (*SHPT*) when the nodes use two different caching schemes. Particularly, the first scheme is the Cache-everything-everywhere caching scheme described in [5], where each item is stored at every intermediate node along the path and each node uses the LRU (Least Recently Used) replacement algorithm in case of an overflow (we call it *En-Route* caching scheme). In the second caching scheme only the node that hosts the requesting subscriber caches an item. Also the LRU replacement algorithm is used within a node in case of an overflow (we call it *Local* caching scheme). The cache hit probabilities of each item at each node for the Local scheme could be retrieved using the closed-form formula presented in [98], but in this chapter we use the discrete event simulator for both strategies in order to compute those probabilities. Without loss of generality we also assume that all caches have the same caching capacity ($C_v = C$, $\forall v \in \mathcal{V}$).

Since no ICN infrastructure has been deployed for commercial use yet, no publicly available data sets exist for performance evaluation. Thus, realistic synthetic workload generators are used instead [99]-[49]. The request rate for an item at each node is determined by its *popularity*. Here we approximate the popularity of the items by a Zipf law of exponents z_{pop}. Literature provides ample evidence that the file popularity in the Internet follows such a distribution [44]-[47]. We denote by $\vartheta_v = \{\vartheta_v^m : m \in \mathcal{M}, v \in \mathcal{V}\}$ the popularity of each item m at node v.

In particular, we consider seven typical values for z_{pop} (popularity exponent of the Zipf distribution) ranging from -1 to 1, i.e. $z_{pop} \in \mathcal{Z} = \{-1, -0.75, -0.5, 0, 0.5, 0.75, 1\}$). A Zipf distribution of negative exponent (e.g. $z_{pop} = -1$) means that out of the M items the most popular item is the M-th, the second most popular is the $(M-1)$-th and so on, with the first item being the least popular. On the other hand a Zipf distribution of positive exponent (e.g. $z_{pop} = 1$) means that the first item is the most popular and the M-th is the least popular. A zero value of $z_{pop} = 0$ corresponds to equally popular items. We assume that in each node a total of 5 requests per second is generated. Thus, the request rate of each item at each node varies from 0-5 reqs/sec according to its popularity.

Generally, the popularity of each item differs from place to place, a phenomenon that is referred to as locality of interest. Thus, we assume that the network is partitioned in $|\mathcal{Z}|$ neighborhoods. Within each neighborhood the popularity of each item m is constant. We assume that the size of each neighborhood follows a Zipf distribution of exponent z_{loc}, i.e. $\lambda_k : k = 1, \ldots, |\mathcal{Z}|$ is the size of partition k, where the popularity of items is given by the corresponding popularity exponent z_{pop}.

In particular, the first partition $k = 1$ consists of $\lfloor \lambda_1 \cdot V \rfloor$ nodes, where the popularity of each item follows a Zipf law of popularity exponent -1. This set of nodes is computed by choosing randomly a central node and its $\lfloor \lambda_\vartheta \cdot V \rfloor - 1$ closest neighbors, by executing a Breadth First Search (as long as a node has not been already assigned to another neighborhood). Note that $z_{loc} = 0$ means that the items are of uniform locality and hence the $|\mathcal{Z}|$ neighborhoods are of equal size ($\frac{V}{|\mathcal{Z}|}$ nodes each). We used network topologies from the Internet Topology Zoo dataset [50], which contains real network topologies from all over the world. Since for a given network size more than one topologies may exist in the dataset, we used all of them for averaging purposes.

We are interested in the following performance metrics:

- The *Total Transportation Cost, TTC* (in $resps \cdot hops/sec$) for each one of the routing schemes for the transportation of all the items to the nodes, where the requests were made. This metric is the mean number of hops that all the requests (and the corresponding responses) will travel in the network until they reach the server or a cache where the requested item is cached.

- The *Server Hit Ratio, SHR* (in $reqs/sec$) for each one of the routing schemes. This metric is the mean number of requests that will finally reach the hosting server of the requested item, since the item could not be found at an intermediate cache.

- The *Average Link Utilization, ALU* (in resps/sec) for each one of the routing schemes. This metric is the average number of responses that travel through each link of the network.

- The *Maximum Link Utilization, MLU* (in resps/sec) for each one of the routing schemes. This metric is the maximum number of responses that travel through the most constrained link ("busiest") of the network. This metric along with the ALU metric is indicative of the load balancing capabilities of each routing scheme.

- The *Average Path Computation Time, APCT* (in sec) is the average time for the computation of a path by the DP algorithm. Particularly, it is the processing time that the computer used in the simulations requires for the computation of a path. This metric is also indicative of the computational complexity of the proposed DP algorithm.

The performance evaluation part consists of six different experiments. Initially, we examine the convergence of the iterative algorithm presented in Section 6.4 for a given network topology size of the Zoo dataset. Next, we consider scenarios that arise from the synthetic workload generator in order to compare the performance of the proposed routing and caching schemes when varying the number of caches V in the network, the cache capacity C of each cache, the exponent of the locality z_{loc} and the exponent of the popularity z_{pop}. In all the above experimental scenarios we assume that the request pattern at each node does not change between the iterations of the iterative algorithm. In the last experiment we examine the performance of the proposed CAWR scheme when the request pattern is not stable between the iterations.

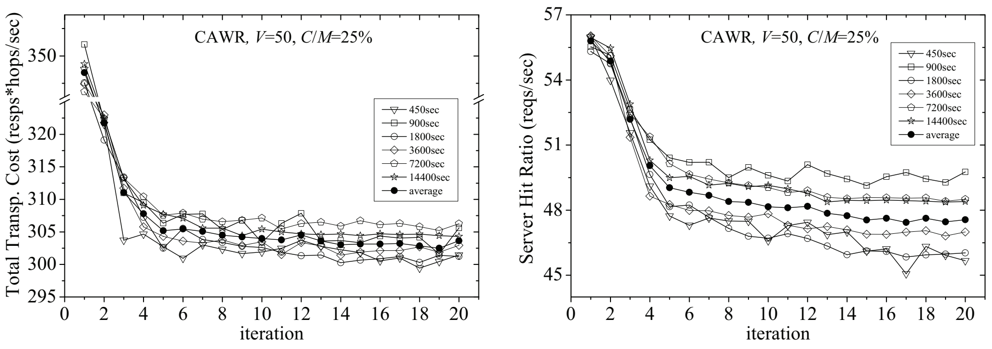

Figure 6.4: Convergence of the cache aware routing algorithm for various observation periods of the discrete event simulator.

Figure 6.4 presents the evolution of the proposed iterative algorithm for a given number of iterations for various values of the observation time period T. It is obvious that the iterative algorithm converges very fast (only after 5 iterations) regardless of the used observation period. Convergence means that after each iteration the DP algorithm computed the same paths computed in the previous iteration. This implies that the cache hit probabilities of the items have also converged and they won't change until the demand pattern changes. Moreover, we observe that the outcome of each different experiment deviates less than 1% from the average value, which implies that the value of the observation period T is of little importance. Of course, in the following experiments we used in our discrete-event simulator a large value for period T and we depict the performance of the network after the convergence of the iterative algorithm.

In Figure 6.5 we depict the impact of the number of nodes/caches V in the network. The capacity of each cache is expressed as the fraction of the items that can be stored in the particular cache ($C_v/M = C/M = 25\%$, $M = 1000$ items). We notice that both the TTC and the SHR performance metrics exhibit a sublinear behavior. Also, the proposed CAWR scheme outperforms the traditional SHPT regardless of the size of the network. Particularly we observe $10\% - 35\%$ ($5\% - 18\%$ when the Local caching scheme is used) improvement regarding the total transportation cost when the En-Route caching scheme is used, even for very small networks where the availability of alternative paths is small, and $15\% - 65\%$ ($10\% - 42\%$ accordingly) improvement regarding the Server Hit Ratio. Particularly, in the case of the SHR when the En-Route scheme is used the results are very impressive, where up to 65% less requests, compared to the SHPT, finally reach the server implying a better utilization of the network resources and minimizing the needs for the deployment of server replicas in the network. This is also evident from the two link utilization metrics where we observe $50\% - 60\%$ decrease of the links' utilization when the the CAWR routing scheme is used instead of the traditional shortest path scheme. Finally, from Figure 6.5 is obvious that the simplistic Local caching scheme outperforms the En-route caching scheme up to 20% regarding TTC (30% regarding SHR) regardless of the used routing scheme. This remark, also observed in the rest of the experiments, further enforces the doubts that have already questioned the cache-everything-everywhere (En-Route)

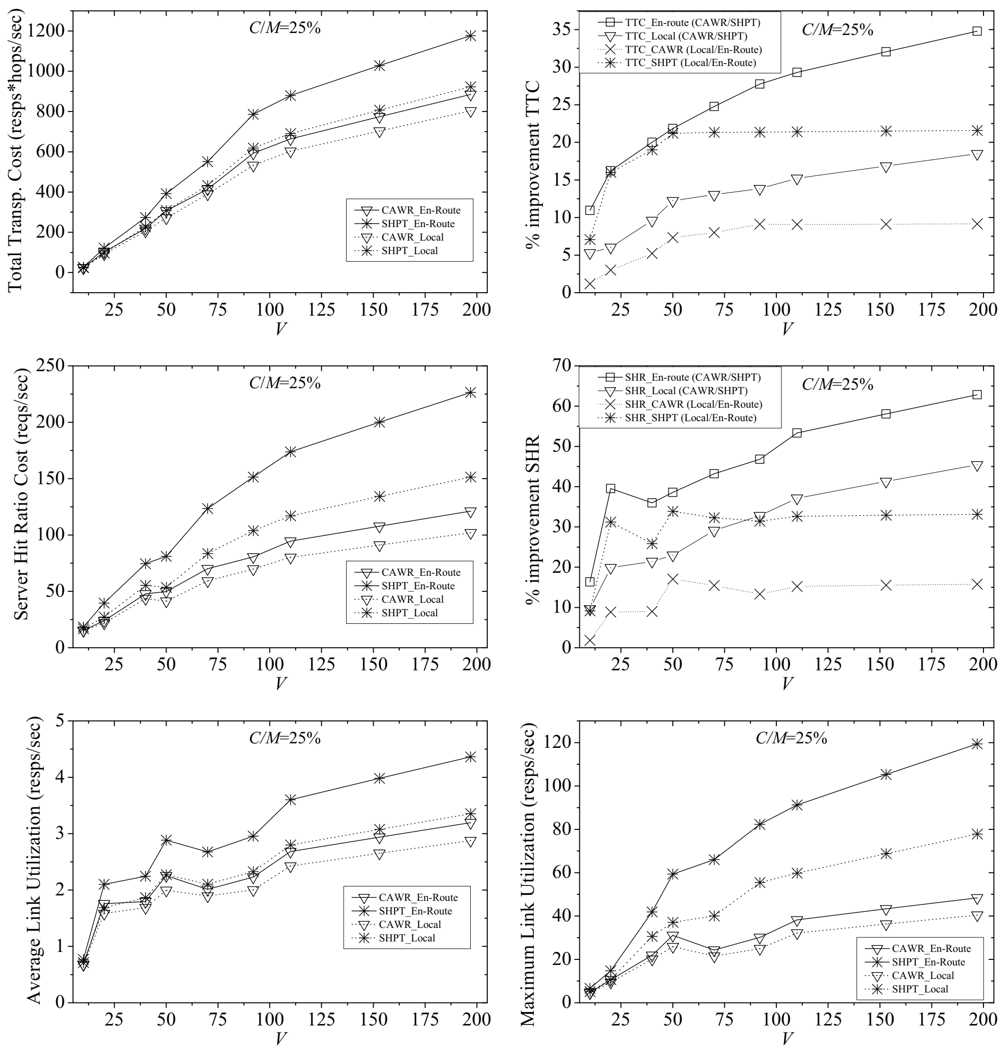

Figure 6.5: The performance of the proposed cache aware routing scheme and the traditional shortest path routing vs. the number V of nodes (caches) in the network, for two different caching strategies.

scheme.

In Figure 6.6 we depict the impact of the cache capacity, expressed as the fraction of the items that can be stored in a cache. In general, we notice similar behavior with the previous experiment, with the proposed CAWR scheme performing better than the SHPT scheme. We also observe that the Local caching scheme outperforms the En-Route scheme, but as expected with the benefit diminishing as we relax the storage capacity constraint and allowing more items to fit in each cache. Finally, in almost every experiment we observed that the SHR improvement is almost twice as much as the improvement in the TTC. This means that even if there are cases where the new routing scheme cannot alleviate the transportation costs, it can at least achieve significantly better utilization of the network resources and reduce the load at the hosting servers. The inherent load balancing capabilities of the proposed CAWR scheme are also evident from the two utilization metrics, where the CAWR

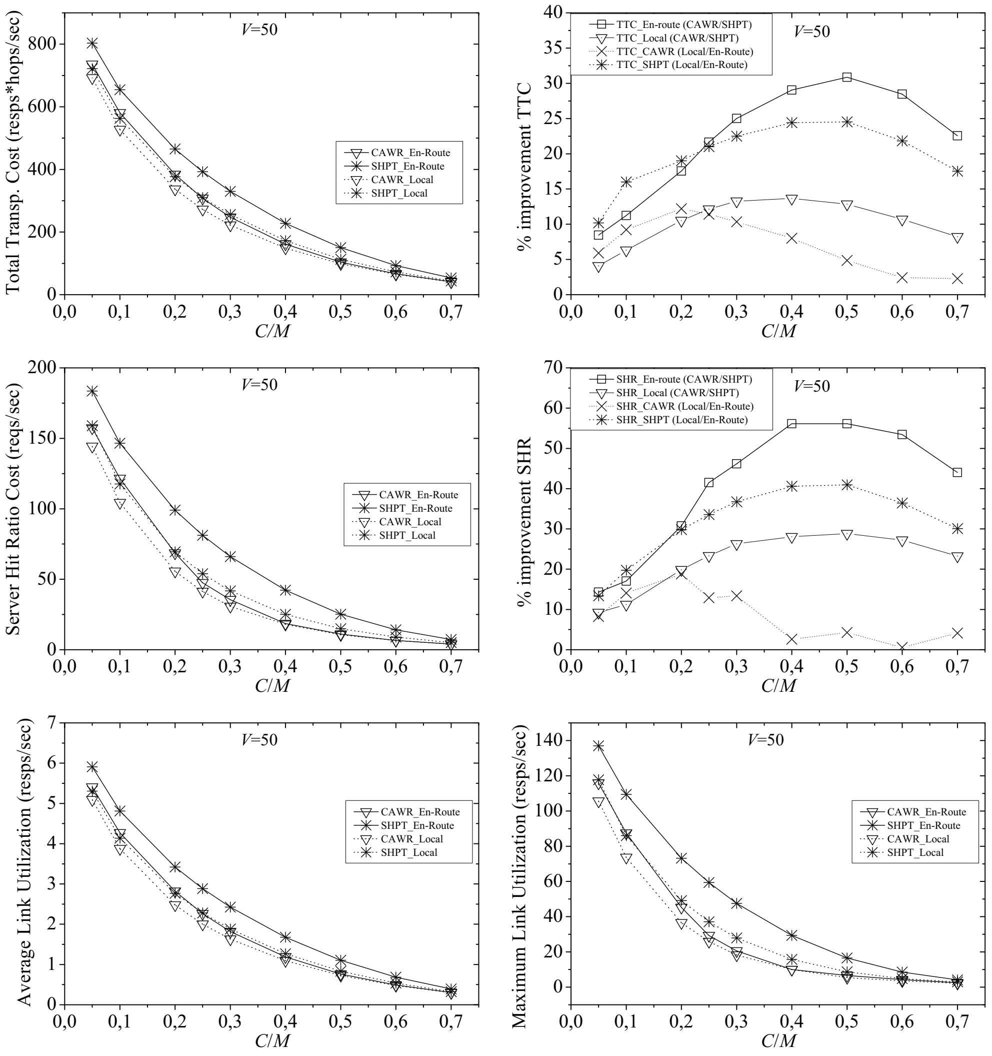

Figure 6.6: The performance of the proposed cache aware routing scheme and the traditional shortest path routing vs. the fraction of the items that can be stored in a cache, for two different caching strategies.

scheme is on average $10\% - 30\%$ better than the SHPT regardless of the underlying caching scheme.

In Figure 6.7 we examine the impact of the locality variations on the performance of the routing schemes. We notice that the proposed CAWR scheme performs better than the SHPT scheme for each one of the metrics. Generally, the changes of the locality exponent cause a domino effect requiring significant reorganization of the routing scheme, since they alter the topology of the demands in the network under consideration and consequently the cache hit probabilities of each item in the network. Of course at the convergence the routing schemes perform identical over the different locality values, implying that the sizes of the neighborhoods, where the popularities are assigned, has limited impact on the performance of the routing schemes and those minor differences are due to the way we choose the nodes for the assignment of the popularities.

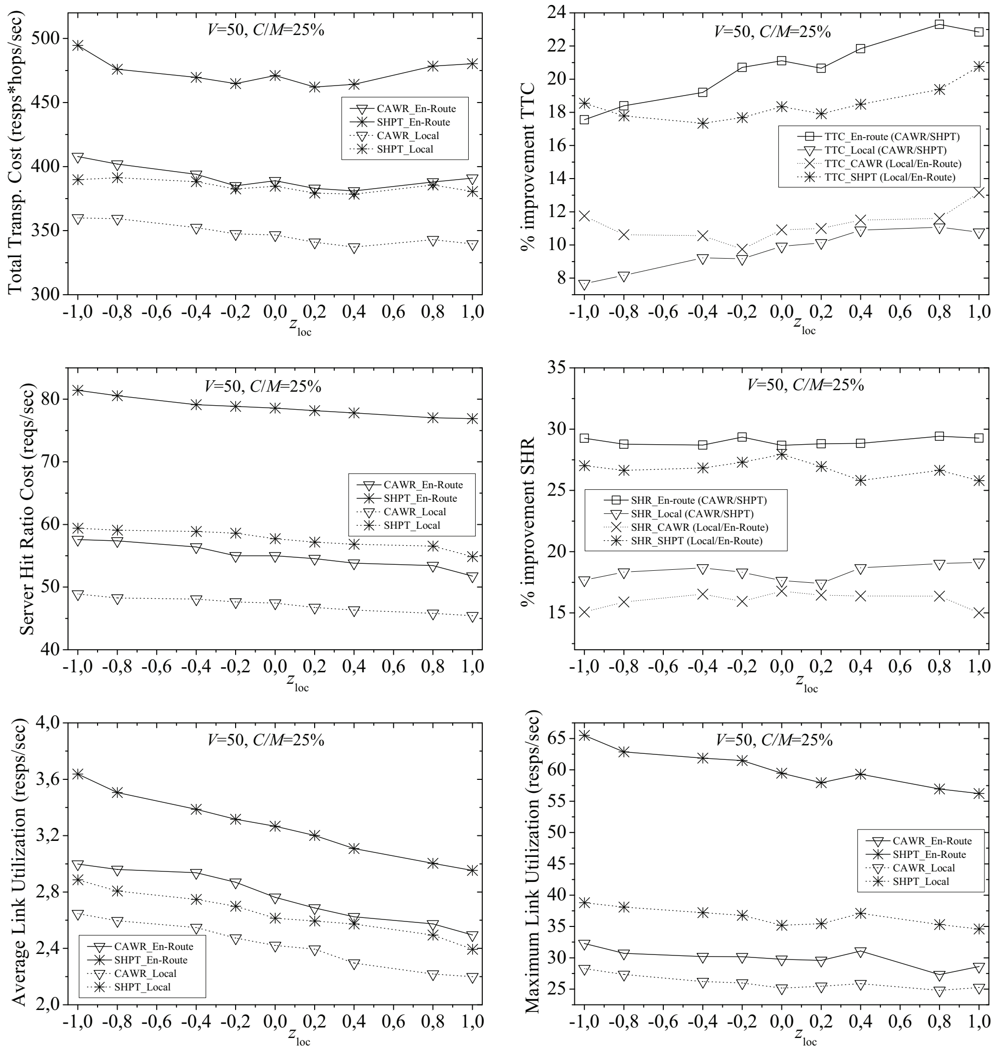

Figure 6.7: The performance of the proposed cache aware routing scheme and the traditional shortest path routing vs. the locality exponent z_{loc}, for two different caching strategies.

In Figure 6.8 we examine the impact of the popularity variations on the performance of the CAWR routing schemes. We also examine its adaptivity when the popularity of the requests change. In order to examine the adaptivity of the proposed routing scheme we initially assume that the popularities assigned to the nodes of the network, using a given locality, are given by the vector $Z = (-1, -0.75, -0.5, 0, 0.5, 0.75, 1)$ and at each different experiment (different points in the figure) this vector changes by a given factor. This factor ranges from 10% to 200%. A change of 10% means that the new vector of popularities is $Z = (-0.9, -0.675, -0.45, 0, 0.45, 0.675, 0.9)$, whereas a change of 100% transforms the vector of popularities to $Z = (0, 0, 0, 0, 0, 0, 0)$ and a change of 200% inverts the vector $Z = (1, 0.75, 0.5, 0, -0.5, -0.75, -1)$. We also depict the performance of the initial routing *CAWR_init* with the new demand pattern. Particularly, we compare the performance of the CAWR after the computation of the new paths to the performance of the CAWR when we use the initial paths

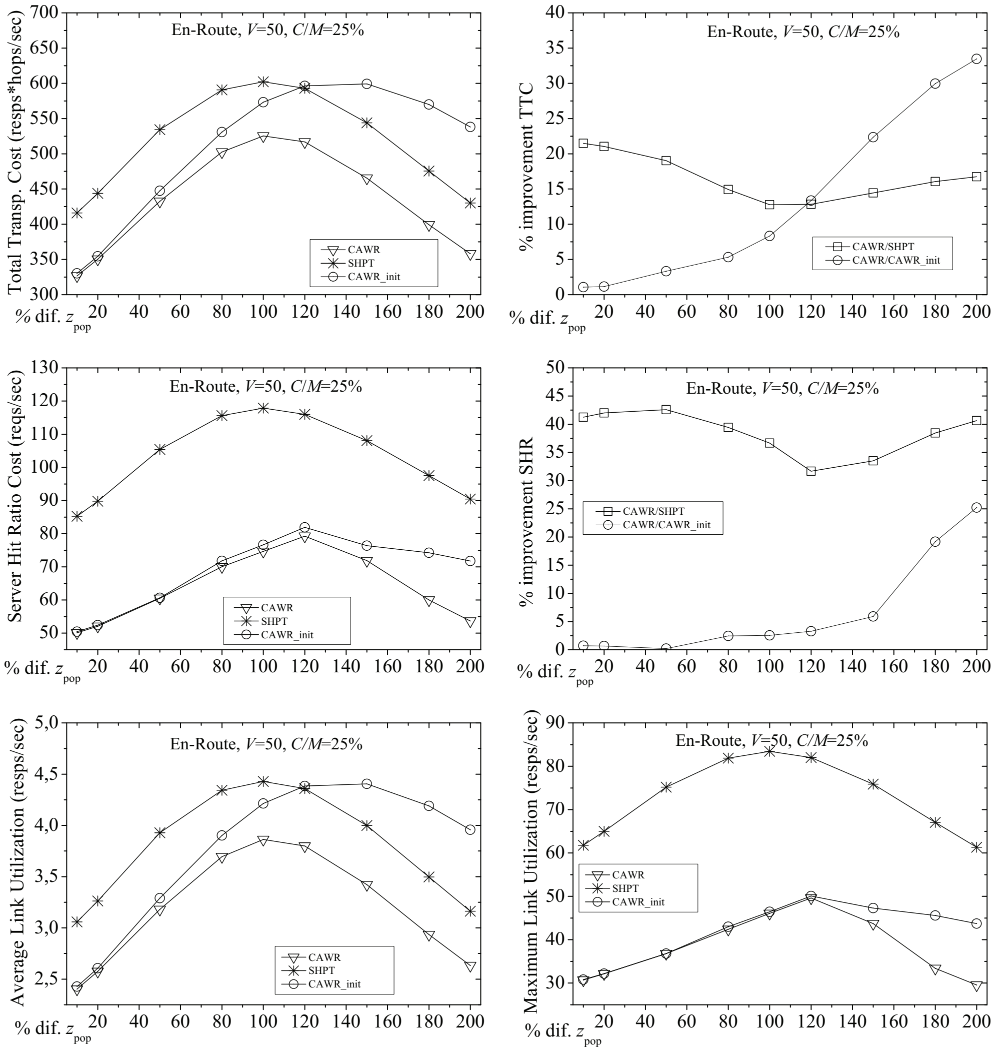

Figure 6.8: The performance of the proposed cache aware routing scheme and the traditional shortest path routing vs. the popularity exponent z_{pop}.

computed for the initial vector of popularities, but under the new demand pattern.

Similarly to the previous experimental scenarios the proposed CAWR scheme performs better than the SHPT scheme for each one of the used metrics. Particularly, we observe $15\% - 20\%$ improvement regarding the total transportation cost and $35\% - 45\%$ improvement regarding the Server Hit Ratio. The interesting in this experimental scenario is in the comparison of the CAWR with the CAWR_init. We observe that when the changing factor of the initial popularities is smaller that 100% the CAWR performs only $1\% - 6\%$ better than the CAWR_init and only when the changing factor is larger than 100% and the popularity vector reverts its sign we observe a difference is the performance up to 35% and 25% regarding the TTC and the SHR metric. This means that as long as the ranking of the items, regarding their popularity, do not change and despite that fact that the items' popularity become more uniform the initial paths are still good enough and even better than the shortest paths.

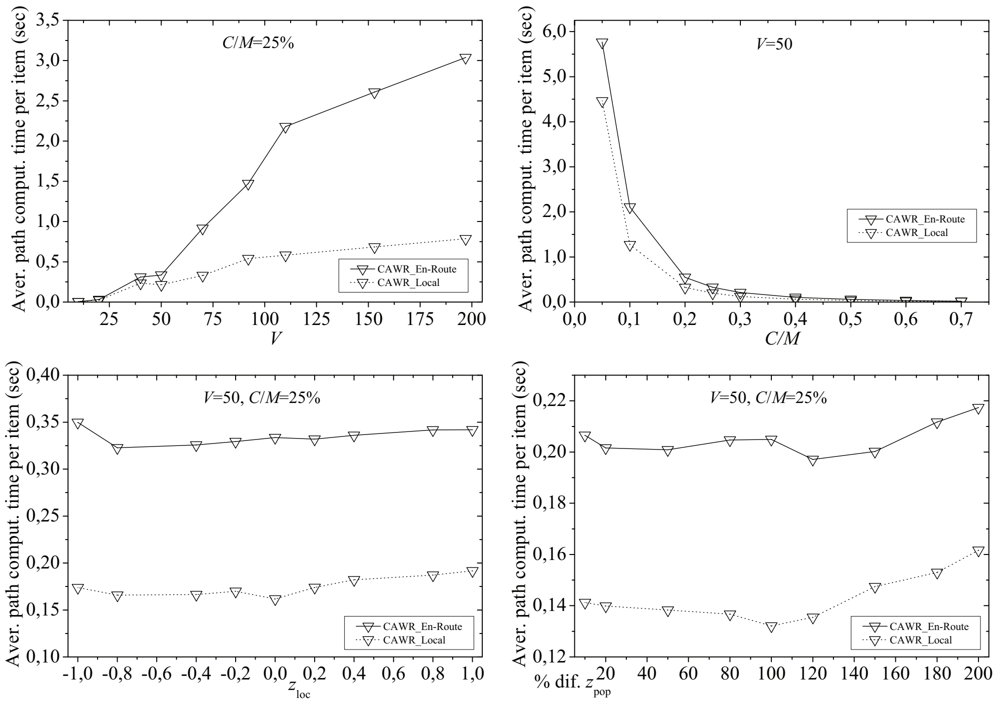

Figure 6.9: Average path computation time per item vs. the number of caches in the network, the cache capacity of each node, the locality and the popularity exponent for two different caching strategies.

Note that we omit the Local caching scheme from Figure 6.8, since it performs similarly to the En-route scheme but being, as in every other experiment, 10% better regarding TTC (20% regarding SHR) than the En-Route scheme.

Figure 6.8 could also be used as a benchmark for the resource managers in their decision to re-compute the minimum transportation cost paths or not upon the detection of a change in the popularity pattern. Particularly, the difference between the TTC and SHR metric produced by the initial routing and the TTC and the SHR metric after the completion of the iterative algorithm enables the RMs to skip or not the changing of the routing scheme. For example when the observed popularities change up to 100% we observe only a $1\% - 7\%$ decrease in the performance, whereas when the observed popularities revert the decrease in the performance is in the area of 30%, meaning that in the first case the RMs could skip the change of the routing scheme, whereas in the second case such a change is crucial.

In Figure 6.9 we depict the average path computation time per item of an average computer (a laptop with a dual-core 1.3 GHz CPU and 2 GB RAM that runs MATLAB for the execution of the DP algorithm). This metric along with the complexity analysis of Section 6.4.1 give pointers for the future applicability of the proposed iterative algorithm and the new cache aware routing scheme. Apart from the size of the network we observe that the APCT metric depends heavily on the cache size of each node, whereas the locality and the popularity exponent has no impact at all. The proposed

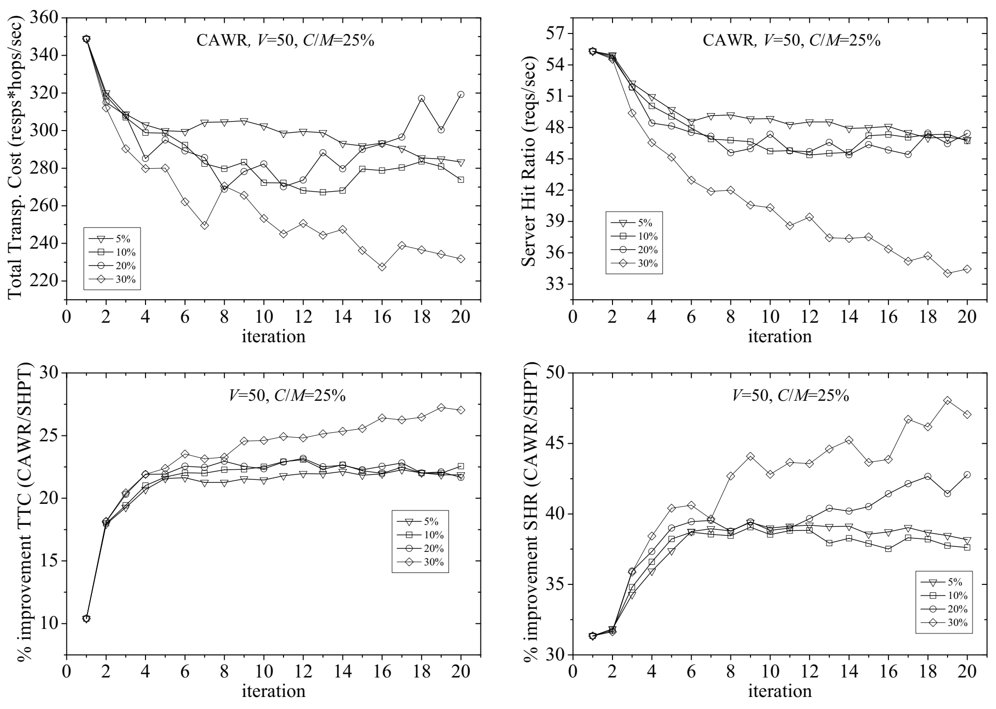

Figure 6.10: The performance of the proposed cache aware routing scheme when the request request patterns are not stable between the iterations of the DP algorithm.

DP algorithm, at each step, adds to the path the link that leads to a node closer to the server and has the largest cache hit probability for the given item. Increasing the cache capacity of each node also increases the cache hit probability of every item. This means that in order to retrieve an item with the minimum transportation cost we follow shorter paths and the DP algorithm terminated faster ("visits" less links in the network).

In all of the above experiments we assumed that the request patterns are stable between the iterations of the iterative algorithm. This is also a necessary condition for the iterative algorithm to converge. In Figure 6.10 we examine the performance of the CAWR routing scheme when the request pattern at each node changes randomly between the iterations of the iterative algorithm up to a given factor $a \in A = \{5\%, 10\%, 20\%, 30\%\}$. Particularly, we use a uniform random value $\Lambda \in [-1, 1]$ and if at iteration itr the exponent of the zipf popularity at a given node $v \in \mathcal{V}$ is $z^v_{pop}(itr)$ at the following iteration $itr+1$ the exponent of the zipf popularity at the same node v is:

$$z^v_{pop}(itr+1) = \begin{cases} z^v_{pop}(itr) \cdot (1+\Lambda) \cdot a, & \text{if } z^v_{pop}(itr) \neq 0 \\ \Lambda \cdot a, & \text{otherwise.} \end{cases} \tag{6.8}$$

From Figure 6.10, it is obvious that the iterative algorithm is not converging when the request pattern at each node of the network is not stable between the iterations, but for small values of the changing factor a ($a \leq 20\%$) the DP algorithm computes up to 95% the same paths between two consecutive iterations. This is actually the reason that for small values of a the performance of the CAWR deviates less than 5% between two different iterations. What is interesting from this experiment is the comparison of the CAWR scheme over the SHPT scheme. It is obvious that by executing no more than two iterations of the iterative algorithm we can achieve at least 18% less traffic in the network and at least 35% less server hits despite the factor a. This means that the proposed CAWR routing scheme is robust and could be applied in highly volatile environments, outperforming the traditional shortest path routing scheme, even if the proposed DP algorithm computes paths with outdated information. Of course, in such a volatile environment a network operator can assume that the algorithm converges when at two consecutive iterations the DP algorithm computes a large number β (e.g. $\beta \geq 0.95$) of the same paths. In that scenario Step 4 of the iterative algorithm presented in Section 6.4 terminates when $P_v(itr+1) \equiv \beta \cdot P_v(itr), \forall v \in \mathcal{V}$.

6.6 Chapter Conclusions

In this chapter, we proposed a new intra-domain cache aware routing scheme for the computation of the paths with the minimum transportation cost in ICN implementations, where in-network opportunistic caching is enabled. We particularly presented a Dynamic programming approach for the computation of the paths when the cache hit probability of each item at each node change based on the observed item request patterns such as their popularity and locality and the used caching scheme. We also presented an iterative algorithm for the cases where the cache hit probability of each item depends on the routing scheme. Finally, we present a resource management architecture for the cache aware routing in an ICN, where distributed resource managers make route decisions, and we validate the proposed approach through extensive simulations. It is evident that the use of alternative paths other than the shortest path (in terms of hops) with different caching capabilities for each item give significant performance benefits and reduce significantly the total transportation cost and the load of the hosting server.

Chapter 7

Conclusions and Future Work

7.1 Summary of the Contributions

In this thesis, we investigated and developed key network management functions for ICN approaches related to route and cache management. Particularly, we developed mechanisms that manage the routing processes by influencing the forwarding of interest/subscription packets and make caching decisions about where and which item to cache as well as influence the cache replacement policies. Our aim was to improve the operations and overall utility of the ICN architectures through extensive and novel usage of caching as an inherent architectural function. Both opportunistic in-network and service-specific managed caching (CDN-like replication) were considered in addressing this challenge.

7.1.1 Replication management

We have presented a three phase framework as a contribution to the problem of information replication in an ICN environment. The objective of the proposed framework was to minimize the total traffic load in the network subject to installing a predefined number of replication devices, and given that each device has storage limitations. The proposed framework is composed by three phases namely the *Planning*, the *Off-line Assignment* and the *On-line Replacement* phase which manage the content and the location of each replication device in the network. In the Planning phase, the proposed framework selects those nodes of the network to place the replication devices while in the Off-line Assignment phase each information item is assigned, based on its popularity, at a subset of the selected replication points so that the targeted objective is satisfied. Finally, the On-line Replacement/Reassignment phase dynamically reassigns information items in the replication devices based on the observed items changing request patterns.

Particularly, we enhanced the CBPS communication paradigm with an advertisement and a request/response mechanism so that replicas can advertise what they have stored and subscribers can retrieve it, while we proposed a new algorithm for the selection of the replication points in the network based on the locality and the popularity of the interests for each information item, the targeted replication degree of each item and the storage capacity (limitation) of each replication device. We further proposed two alternative mechanisms for the off-line assignment of the replicas of each item among the selected replication points. Evaluation via simulations of the performance of the system regarding the clients response latency and the network traffic shows that our planning and off-line replica assignment scheme is a promising solution in almost any scenario.

We also proposed an autonomic cache management architecture that dynamically reassigns information items to the caches of an ICN approach. In particular, we proposed four distributed online cache management algorithms requiring different levels of cooperation among the autonomic managers and we compare them in terms of their performance, complexity, message overhead and convergence time. We provided also a method to calculate a lower bound of overall network traffic cost, for distance-regular network topologies. Our numerical results provide evidence that network wide knowledge and cooperation give significant performance benefits and reduce the time to convergence at the cost of additional message exchanges and computational effort. In more details the cooperative algorithm provides the best performance regarding overall network traffic, but requires a high level of cooperation among the managers and hence is of very high computational and communication complexity. On the other hand, the two holistic algorithms perform close to the cooperative, but converge in a fraction of the iterations required by the cooperative. Finally, the myopic algorithm requires the least cooperation and hence is appropriate for larger network setups, but its performance is significantly worse that the rest. Thus, the proposed analysis may serve as a valuable tool for the network manager so as to select the most appropriate algorithm for his needs, depending on specific network parameters (e.g. network size, number of information items, volatility of the request pattern). The proposed three phase replication framework is generic so that it can apply in almost every ICN proposed architecture.

7.1.2 In-network opportunistic caching

We have described our design and implementation of an opportunistic caching mechanism for peer-to-peer ICN approach, where servers do not exist. The proposed opportunistic caching mechanism aims also at preserving the information over time instead of only making information available in nearer space as in traditional caching schemes. Particularly, we enhanced the CBPS architectural design with a request/response scheme so that subscribers can retrieve cached information/data from other nodes in the network, assuming that each network node has a limited cache and there are no servers in the network. We also proposed two duplicate preventing mechanisms, that will handle the possible production of multiple identical responses to a request, due to the multiple caching of information/data at different nodes. We have also decomposed the caching mechanism in a set of basic policies/strategies, present at each set the most known and traditional policies and propose an information-centric policy at each one of them. Additionally, we have proposes a stochastic model that captures the dynamics of the newly proposed policies and we described a prototype implementation of the proposed opportunistic caching mechanism that was evaluate it through simulations and Planetlab experimentation. Finally, we presented a modification to the proposed caching mechanism to enable mobility of the subscribers. Both the evaluation through simulations and the system prototype experimentation in PlanetLab shows that the proposed caching mechanism outperforms traditional caching mechanisms retaining at the same time the traffic overhead in low levels.

7.1.3 Cache aware routing

We proposed a new cache aware intra-domain routing scheme for both centralized and distributed ICN architectures, that dynamically computes the routes followed by each subscription/interest for each item and from each node in the network. Particularly, we presented a dynamic programming (DP) approach for the computation of the minimum transportation cost paths based on the observed item request patterns, such as their popularity and locality and the used caching scheme, in order to minimize the overall transportation cost imposed by the user requests. Moreover, we propose an iterative algorithm for the computation of the minimum cost transportation paths for those scenarios where the routing decisions interact with the caching decisions. The validation of the proposed routing scheme through simulations revealed that the usage of alternative paths other than the shortest path (in terms of hops) with different caching capabilities for each item give significant performance benefits and reduce significantly the total transportation cost and the load of the hosting server.

7.2 Future Work

The core work presented in this thesis can be extended in many ways such as optimizing different objectives to serve different QoS metrics and SLAs among the storage providers and the content providers. Also it would be interesting, as future work, to explore enhancements to the proposed on-line replacement algorithms that would also take into consideration the cost of replacing the items at the replication points of the network, as well as the processing load of each replacement component when assigning items to them. The work in opportunistic caching can also be extended in many ways, from deriving applications to combining the proposed mechanism with permanent storages/servers. Finally, it would be interesting to explore enhancements to the proposed routing scheme that would also take into consideration the presence of multiple server replicas.

In the area of management of ICN approaches we believe that crucial questions remain and should be considered as future work. Particularly, security/anomaly detection as well as support of seamless mobility and energy efficient use of the ICN resources have not been considered yet. Solutions to mitigate attacks and detect anomalies are prerequisites for network operators to trust the ICN technology. ICN can prevent some security attacks, such as man-in-the-middle attacks or address spoofing, via its native design (i.e., no end-to-end connection between two end-users, no location-based addresses in the ICN implementation, built-in content authentication). But even if security has been taken into consideration from the beginning, many network security aspects are still not yet addressed. For example, some components of a CCN node architecture, such as the Pending Interest Table (PIT) or the Content Store (CS) or the Rendezvous Nodes (RVs) in the PSIRP/PURSUIT architecture are vulnerable to denial of service attacks, e.g., by malicious users sending many simultaneous subscriptions/interests for content that does not exist in the network which will completely fill the nodes' PIT tables or the RV with useless subscriptions (content "poisoning"). It would be interesting to identify which attacks are possible in a ICN environment and investigate solutions to mitigate them. These solutions can be based on network behavior analysis offered by generic management modules to detect bursts of interest packets which are possibly part of an attack. Moreover, cooperation between nodes to cache particular content will reduce the impact of a possible attack by preventing nodes from storing the same content.

Another interesting extension of the proposed work would be the identification and analysis of user and content mobility patterns. Considerable research has been dedicated to understand and model the behaviour of mobile users from the social and technological perspectives. Most existing work on predicting mobility patterns, attachment points and connectivity durations aims to minimize periods of disconnection. With ICN however, content can "follow" mobile users, thus achieving shorter transaction periods, therefore this work needs to be re-evaluated.

It is interesting not only to consider user-mobility patterns, but also to extend it to cover group mobility (e.g., in a VAN), as well as content-mobility. The outcome will be monitoring mechanisms for user, group and content mobility, as well as algorithms for predicting the future connectivity points of users, groups and content in order to move or migrate content accordingly. The inherent support of in-network caching provided by ICN could be exploited to identify strategic caching points at the edges of the network. The identification of these points should be based on: i) route selection and transmission scheduling and tradeoffs between energy, delay and cost. For example, caching content closer to the user reduces energy consumption and delivery delay, as seen from the end-users point of view, as well as reducing core-network traffic, as seen from the operators point of view, but it also increases caching deployment costs for the operator. Based on the investigation of such tradeoffs, the outcome could be: i) the identification of strategic caching points close to the user, ii) caching strategies to increase the amount of time content stays in the cache, iii) the corresponding effect to the route selection and transmission scheduling algorithms.

In ICN, mobile nodes can select an access point by taking into account more information than just signal strength. By making each access point aware of its associated mobile nodes interests, this knowledge can be exploited when new nodes join the network or are in the process of a handover. Associating nodes with similar interests within an access point will lead to better utilization of network and device-specific resources and also increase the probability of retrieving the content they are interested in from a nearby cache or, even, from another node in the group. Furthermore, better utilization of resources can be achieved by sophisticated content multiplexing in caches, which can lead to an optimized distribution of both the content and the mobile nodes associated to the network. This procedure can guarantee the efficient utilization of network resources, since overload or underload situations will be avoided and load balancing will take place in a self-managed way between mobile and access network nodes.

Finally, another interesting task would be to investigate strategies and mechanisms to reduce energy consumption in current and future ICN approaches. Energy efficiency should be considered from the design phase, based on the tradeoffs between energy, delay and cost. Considering network operation, new solutions should be investigated to balance costs and QoS constraints. In contrast to past research, which focuses on routing per se, it is interesting to investigate route selection, which incorporates selecting both a content source and a path towards it. The tradeoffs between energy, delay and cost should be studied both from the end-users and from the operators perspective, in order to achieve better network performance and more cost-effective network operation. Mobile node interests for content can be utilized to provide better network performance in terms of throughput, end-to-end delay and energy consumption both in wireless and wired access networks.

Bibliography

[1] S. Carew, “Users complain, AT&T blames data tsunami,” Reuters Media File, Feb. 14, 2012.

[2] Cisco Systems, “Cisco visual networking index: Forecast and methodology, 2011-2016” 30 May 2012.http://tinyurl.com/VNI2011

[3] http://www.intel.com/content/www/us/en/communications/internet-minute-infographic.html

[4] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, “A Data-Oriented (and Beyond) Network Architecture,” In Proc. of SIGCOMM, 2007.

[5] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. Briggs, R. Braynard, “Networking named content,” ACM CoNEXT, Rome, Italy, Dec. 2009.

[6] A. Detti, N. Blefari-Melazzi, S. Salsano, and M. Pomposini, “CONET: A Content Centric Inter-Networking Architecture,” in ACM SIGCOMM workshop in Information-centric networking, 2011.

[7] PURSUIT project, available at http://www.fp7-pursuit.eu, 2011.

[8] D. Lagutin, K. Visala, and S. Tarkoma, “Valencia FIA book 2010 Publish/Subscribe for Internet: PSIRP Perspective,” IOS Press, 2010.

[9] SAIL FP7 EU project, http://www.sail-project.eu/.

[10] The IETF DECADE Working Group, 2012, available at:
https://datatracker.ietf.org/wg/decade/charter/

[11] Object Management Group. “CORBAservices: Common Object Service Specification,” Technical report, Object Management Group, July 1998.

[12] TIBCO. “TIB/Rendezvous,” White paper, TIBCO, Palo Alto, CA.1999.

[13] Named Data Networking project, available at http://named-data.net, 2011.

[14] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley and T. D. Chandra, “Matching events in a content-based subscription system,” In Proc. of 18th ACM PODC Atlanta, May, 1999.

[15] A. Carzaniga, D. Rosenblum and A. Wolf, “Design and evaluation of a wide-area event notification service,” ACM Transactions On Computer Systems, vol. 19, pp. 332–383, 2001.

[16] B. Segall and D. Arnold, “Elvin has left the building: A publish/subscribe notification service with quenching,” In Proc. of AUUG, Brisbane, Australia, Sept. 3-5, pp. 243–255, 1997.

[17] G. Cugola and G. Picco, ”REDS, A Reconfigurable Dispatching System,” in 6th International workshop on Software Engineering and Middleware, pp. 9-16, Oregon, 2006.

[18] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan and James Wilcox, “Information-centric networking: seeing the forest for the trees,” 10th ACM Workshop on Hot Topics in Networks, p.1-6, November 14-15, 2011, Cambridge, Massachusetts.

[19] B. Li, M. J. Golin, G. F. Ialiano and X. Deng, “On the Optimal Placement of Web Proxies in the Internet,” In Proc. of INFOCOM, March 1999.

[20] I. Cidon, S. Kutten, R. Soffer, “Optimal allocation of electronic content,” In Proc. of INFOCOM, Anchorage, April 2001.

[21] J. Kangasharju, J. Roberts, K. Ross, “Object replication strategies in content distribution networks,” Comput. Commun. Elsevier, vol. 25, pp. 376–383, March 2002.

[22] L. Qiu, V.N. Padmanabhan and G. Voelker, “On the placement of web server replicas,” In Proc. of IEEE INFOCOM, Anchorage, USA, Apr. 2001.

[23] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala and V. Pandit, “Local search heuristics for k-median and facility location problems,” In Proc. of 33rd ACM Symp. on Theory of Computing, 2001.

[24] M. Charikar and S. Guha, “Improved combinatorial algorithms for facility location and k-median problems,” In Proc. of the 40th Annual IEEE Symp. on Foundations of Computer Science, pp. 378-388, Oct. 1999.

[25] M. Charikar, S. Khuller, D. Mount, and G. Narasimhan, “Facility location with outliers,” In Proc. of the 12th Annual ACM-SIAM Symp. on Discrete Algorithms, Washington DC, Jan. 2001.

[26] D.B. Shmoys, E. Tardos and K.I. Aardal, “Approximation algorithms for facility location problems,” In Proc. of the 29th Annual ACM Symp. on Theory of Computing, pp. 265-274, 1997.

[27] E. Cronin, S. Jamin, C. Jin, T. Kurc, D. Raz and Y. Shavitt, “Constrained mirror placement on the Internet,” in IEEE JSAC, 36(2), Sept. 2002.

[28] N. Laoutaris, O. Telelis, V. Zissimopoulos, and I. Stavrakakis, ”Distributed selfish replication,” IEEE TPDS, vol. 17, no. 12, 2006.

[29] S. Zaman, D. Grosu, ”A Distributed Algorithm for the Replica Placement Problem,” IEEE TPDS, Jan. 2011.

[30] S. Borst, V. Gupta, A. Walid, ”Distributed Caching Algorithms for Content Distribution Networks”, in IEEE INOFCOM, San Diego, USA, March 2010.

[31] M. Karlsson, Ch. Karamanolis and M. Mahalingam, “A Framework for Evaluating Replica Placement Algorithms”, http://www.hpl.hp.com/ techreports/2002/HPL-2002-21, 2002.

[32] Li G., Cheung A., Hou S., Hu S., Muthusamy V., Sherafat R., Wun A., Jacobsen H., and Manovski S., “Historic data access in publish/subscribe,” In Proc. of DEBS, pp. 80–84, Toronto, Canada, 2007.

[33] W. Jiang, R. Zhang-Shen, J. Rexford, and M. Chiang, Cooperative content distribution and traffic engineering in an isp network, in Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems, ser. SIGMETRICS 09, 2009, pp. 239250.

[34] B. Frank, I. Poese, G. Smaragdakis, S. Uhlig, and A. Feldmann, "Content-aware traffic engineering," in Proceedings of the ACM SIGMET- RICS/PERFORMANCE 2012 joint international conference on Measurement and Modeling of Computer Systems, 2012, pp. 413414.

[35] N. Kamiyama, T. Mori, R. Kawahara, S. Harada, and H. Hasegawa, "Isp- operated cdn," in Proceedings of the 28th IEEE international conference on Computer Communications Workshops, ser. INFOCOM09, 2009, pp. 4954.

[36] K. Cho, H. Jung, M. Lee, D. Ko, T. Kwon, and Y. Choi, "How can an isp merge with a cdn?' Communications Magazine, IEEE, vol. 49, no. 10, pp. 156162, oct. 2011.

[37] R. Cohen, L. Katzir and D. Raz, "An Efficient Approximation for the Generalized Assignment Problem," Information Processing Letters, Vol. 100, pp. 162-166, Nov. 2006.

[38] G. B. Dantzig, "Discrete-Variable Extremum Problems," Operations Research Vol. 5, No. 2, April, pp. 266-288, 1957.

[39] D. P. Palomar, M. Chiang, " A tutorial on decomposition methods for network utility maximization," IEEE JSAC, 24(8), pp. 1439–1451, 2006.

[40] A. Rouskov, D. Wessels, "Cache Digest," 3rd Inter. WWW caching workshop, June 1998.

[41] L. Fan, P. Cao, J. Almeida, A. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," ACM SIGCOMM, pp. 254–265, Feb. 1998.

[42] A.E. Brouwer, A.M. Cohen, A. Neumaier (1989), "Distance Regular Graphs," New York: Springer-Verlag.

[43] http://mathworld.wolfram.com/Distance-RegularGraph.html

[44] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, "Web caching and Zipf-like distributions: evidence and implications," IEEE INFOCOM, NY, March 1999.

[45] V. N. Padmanabhan, L. Qiu, " The content and access dynamics of a busy wed site," ACM SIGCOMM, Stockholm, Sweden, Aug. 2000.

[46] M. E. J. Newman, "Power laws, pareto distributions and Zipfs law," Contemporary Physics, vol. 46, pp. 323-351, 2005.

[47] L. A. Adamic and B. A. Huberman, "Zipfs law and the Internet," Glottometrics, vol. 3, pp. 143150, 2002.

[48] S. Tarkoma, J. Kangasharju, "Optimizing content-based routers: posets and forests," Distributed Computing, vol. 19, Springer, pp. 62-77, 2006.

[49] A. Majumder, N. Shrivastava, R. Rastogi and A. Srinivasan,"Scalable content-based routing in Pub/Sub systems," IEEE INFOCOM, pp. 567-575, 2009.

[50] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden and M. Roughan, "The Internet Topology Zoo," IEEE JSAC, vol. 29, no. 9, Oct 2011.

[51] T. Wauters, J. Coppens, F. D. Turck, B. Dhoedt and P. Demeester, “Replica placement in ring based content delivery networks,” Elsevier Computer Communications, vol. 29, pp. 3313-3326, 2006.

[52] L. Qiu, V. Padmanabhan, G. Voelker, ”On the placement of web server replicas,” IEEE INFOCOM, pp. 15871596, 2001.

[53] D. Wessels, K. Claffy, “Applications of Internet Cache Protocol (ICP), v.2,” ITF, May 1997.

[54] P. Vixie, D. Wessels, “RFC 2756: Hyper Text Caching Protocol,” Jan. 2000.

[55] P. Rodriguez, C. Spanner, E.W. Biersack, “Analysis of Web Caching Architectures: Hierarchical and Distributed Caching,” in ACM Trans. on Networking, Aug. 2001.

[56] M. Pitkanen and J. Ott, “Enabling Opportunistic Storage for Mobile DTNs,” in Elsevier, Pervasive and Mobile Computing Vol. 4, pp. 579-594, Oct. 2008.

[57] A. Anand, A. Gupta, A. Akella, S. Seshan and S. Shenker, “Packet caches on routers: the implications of universal redundant traffic elimination,” in SIGCOMM Comput. Commun. Rev., Oct. 2008.

[58] P. Srebrny, T. Plagemann, V. Goebel, A. Mauthe, “CacheCast: Eliminating Redundant Link Traffic for Single Source Multiple Destination Transfers,” in 2010 International Conference on Distributed Computing Systems.

[59] Y. Zhu, M. Chen and A. Nakao, “CONIC: Content-Oriented Network with Indexed Caching,” in INFOCOM IEEE Conference on Computer Communications Workshops, pp.1-6, 15-19 March 2010.

[60] Y. Chen, et al., “Efficient and adaptive Web replication using content clustering,” IEEE Journal on Selected Areas in Communications 21, 6 (Aug.2003), 979–994.

[61] N. Fujita, Y. Ishikawa, A. Iwata, R. Izmailov, “Coarse-grain replica management strategies for dynamic replication of Web contents,” Computer Networks 45, (2004), 19–34.

[62] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, “Summary cache: a scalable wide-area web cache sharing protocol,” IEEE/ACM Trans. Netw., vol. 8, pp. 281–293, June 2000.

[63] A. Dan and D. Towsley, “An approximate analysis of the lru and fifo buffer replacement schemes,” ser. SIGMETRICS, 1990, pp. 143–152.

[64] P. R. Jelenkovic , A. Radovanovic , and M. S. Squillante, “Critical sizing of lru caches with dependent requests,” Journal of Applied Probability, vol. 43, no. 4, pp. 1013–1027, 2006.

[65] H. Che, Z. Wang, and Y. Tung, “Analysis and Design of Hierarchical Web Caching Systems,” in IEEE INFOCOM, 2001, pp. 1416–1424.

[66] S. Srikantaiah, E. Kultursay, T. Zhang, M. T. Kandemir, M. J. Irwin and Y. Xie, “Morphcache:A reconfigurable adaptive multi-level cache hierarchy,” HPCA 2011, pp. 231–242.

[67] N. Laoutaris, G. Smaragdakis, A. Bestavros, I. Matta and I. Stavrakakis, “Distributed Selfish Caching,” IEEE TPDS, vol. 18, no. 10, pp. 1361–1376, October 2007.

[68] E. J. Rosensweig and J. Kurose, “Breadcrumbs: Efficient, Best-Effort Content Location in Cache Networks,” in INFOCOM, 2009.

[69] D. Perino and M. Varvello, “A reality check for content centric networking,” in ACM SIGCOMM ICN Workshop, 2011, pp. 4449.

[70] S. Arianfar, P. Nikander, and J. Ott, “On content-centric router design and implications,” in ReArch Workshop, vol. 9. ACM, 2010, p. 5.

[71] W. K. Chai, I. Psaras and G. Pavlou, “Cache Less for More In Information-Centric Networks,” IFIP NETWORKING 2012, Prague, Czech Republic, May 2012.

[72] S. Wang, J. Bi, J. Wu, Z. Li, W. Zhang and X. Yang, “Could in-network caching benefit information-centric networking?,” in AINTEC 2011.

[73] Z. Li and G. Simon, “Time-Shifted TV in Content Centric Networks: the Case for Cooperative In-Network Caching,” in IEEE ICC 2011.

[74] C. Fricker, P. Robert, J. Roberts and N. Sbihi, “Impact of traffic mix on caching performance in a content-centric network,” in IEEE NOMEN 2012.

[75] R. Chand and A. Felber, “A scalable protocol for content-based routing in overlay networks,” in 2nd IEEE International Symp. on Network Computing and Applications, 2003.

[76] J. Wang, “A survey of web caching schemes for the Internet,” in ACM SIGCOMM Computer Communication Review, pp. 36-46, 1999.

[77] S. U. Khan and I. Ahmad, “Comparison and analysis of ten static heuristics-based Internet data replication techniques,” in Journal of Parallel and Distributed Computing, vol. 68, pp. 113-136, 2008.

[78] P. Padmanabhan, L. Gruenwald,, A. Vallur and M. Atiquzzaman, “A survey of data replication techniques for mobile ad hoc network databases,” in VLDB Journal, vol. 17, pp. 1143-1164, 2008.

[79] J. Ardelius, B. Grönvall, L. Westberg and Å. Arvidsson, “On the effects of caching in access aggregation networks,” In ACM workshop on Information-centric networking, pp. 67-72, 2012.

[80] H. Che, Y. Tung, and Z. Wang, “Hierarchical web caching systems: Modeling, design and experimental results,” in IEEE JSAC, pp. 1305-1314, 2002.

[81] A. Dan and D. Towsley, “An approximate analysis of the LRU and FIFO buffer replacement schemes,” IN ACM SIGMETRICS Performance Evaluation Review, pp. 143-152, 1990.

[82] C. Fricker, P. Robert, and J. Roberts, “A versatile and accurate approximation for LRU cache performance,” Arxiv preprint arXiv:1202.3974, 2012.

[83] Latouche G., Ramaswami V., “Introduction to Matrix Analytic Methods in Stochastic Modeling,” SIAM, Philadelphia, 1999.

[84] Neuts M., “Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach”, Johns Hopkins, 1994.

[85] http://www.planet-lab.org/

[86] Cugola G., Nitto E.D. and Fugetta A., “The Jedi event-based infrastructure and its application to the development of the opss wfms,” IEEE Trans. Softw. Eng. 27, 9 (Sept.), pp. 827–850, 2001.

[87] M. Caporuscio, A. Carzaniga, A. L. Wolf, “Design and evaluation of a support service for mobile, wireless publish/subscribe applications,” in IEEE Transactions on Software Engineering, Vol. 29, pp. 1059- 1071, 2003.

[88] L. Mottola, G. Cugola, G.P. Picco, “A Self-Repairing Tree Topology Enabling Content-Based Routing in Mobile Ad Hoc Networks,”in IEEE Transactions on Mobile Computing, Vol. 7, pp. 946-960, 2008.

[89] S. Dibenedetto, C. Papadopoulos and D. Massey, “Routing Policies in Named Data Networking,” in ACM SIGCOMM ICN workshop, 2011.

[90] H. Liu, X. De Foy and D. Zhang, “A Multi-Level DHT Routing Framework with Aggregation,” in ACM SIGCOMM ICN workshop, 2012.

[91] S. Eum, K. Nakauchi, M. Murata, Y. Shoji and N. Nishinaga, “CATT: Potential Based Routing with Content Caching for ICN,” in ACM SIGCOMM ICN workshop, 2012.

[92] V. Lenders, M. May and B. Plattner, “Service discovery in mobile ad hoc networks: A field theoretic approach,” inPervasive and Mobile Computing, vol. 3, pp. 343–370, 2005.

[93] T. Janaszka, D. Bursztynowski and M. Dzida, “On popularity-based load balancing in content networks,” in 24 International Teletraffic Congress, pp. 1–8, 2012.

[94] E. Rosensweig and J. Kurose, “Breadcrumbs: efficient, best-effort content location in cache networks,” in INFOCOM (Mini-Conference), pp. 2631–2635, 2009.

[95] L. Saino, I. Psaras and G. Pavlou, “Hash-routing Schemes for Information-Centric Networking,” to appear in ACM SIGCOMM ICN workshop, 2013.

[96] K. W. Ross, “Hash routing for collections of shared web caches,” Networking Magazine of Global Internetworking, pp. 37–44, 1997.

[97] S. Saha, A. Lukyanenko, A Ylä-Jääski, “Cooperative Caching through Routing Control in Information-Centric Networks,” in Infocom (Mini-Conference), 2013.

[98] N. Laoutaris, “A Closed-Form Method for LRU Replacement under Generalized Power-Law Demand,” arXiv:0705.1970v1, 2007.

[99] A. Carzaniga, M.J. Rutherford and A.L. Wolf, “A routing scheme for content-based networking,” IEEE INFOCOM 2004, pp. 918-928, 2004.

[100] F. Cao and J.P. Singh, “Efficient event routing in content-based publishsubscribe service networks,” Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, IEEE INFOCOM 2004, pp. 929940, 2004.