A Survey of Caching Policies and Forwarding Mechanisms in Information-Centric Networking

Andriana Ioannou and Stefan Weber, *Member, IEEE*

***Abstract*— Information-centric networking (ICN), an alternative to the host-centric model of the current Internet infrastructure, focuses on the distribution and retrieval of content instead of the transfer of information between specific endpoints. In order to achieve this, ICN is based on the paradigm of publish–subscribe and the concepts of naming and in-network caching. Current approaches to ICN employ caches within networks to minimize the latency of information retrieval. Content may be distributed either in caches along the delivery path(s), on-path caching or in any cache within a network, off-path caching. While approaches to off-path caching are comparable to traditional approaches for content replication and Web caching, approaches to on-path caching are specific to the ICN area. The purpose of this paper is to provide a review of the caching problem in ICN, with a focus on on-path caching. To this end, a detailed analysis of the existing caching policies and forwarding mechanisms that complement these policies is given. A number of criteria such as the caching model and level of operation and the evaluation parameters used in the evaluation of the existing caching policies are being employed to derive a taxonomy for on-path caching and highlight the trends and evaluation issues in this area. A discussion driven by the advantages and disadvantages of the existing caching policies and the challenges and open questions in on-path caching is finally being held.**

***Index Terms*—Distributed network architectures, future Internet architectures, information-centric networking, caching technologies, content replication, off-path caching, on-path caching, replica exploitation, forwarding mechanisms.**

I. Introduction

THE ORIGINAL design of protocols for the Internet focussed on the end-to-end communication model for the exchange of information between two participants: a client and a server. However, usage patterns and technologies have changed since these protocols were developed and today's Internet usage is dominated by the distribution and retrieval of content instead of the connection to a specific server. This mismatch of traditional protocols and current usage patterns results in a number of difficulties with regard to availability, mobility, multi-homing, scalability and performance [1], [2].

Manuscript received March 20, 2015; revised November 8, 2015 and February 25, 2016; accepted April 8, 2016. Date of publication May 10, 2016; date of current version November 18, 2016. This work was supported in part by the Higher Education Authority under the Programme for Research in Third-Level Institutions Cycle 5, and in part by the European Regional Development Fund.

The authors are with the School of Computer Science and Statistics, Trinity College Dublin, Dublin 2, Ireland (e-mail: ioannoa@scss.tcd.ie; sweber@scss.tcd.ie).

Digital Object Identifier 10.1109/COMST.2016.2565541

The gap between the original protocols and their current use is bound to increase as reports on the usage patterns of the Internet [3] predict a continuous growth of applications such as video streaming and photo sharing. According to a recent technical paper by CISCO [4], the annual traffic of the Internet is expected to exceed the size of 1.4 zettabytes by the end of 2017 with almost 80% of the traffic being video traffic.

Content Delivery Networks (CDNs) [5] and Peer-to-Peer (P2P) [6] networks are considered to be the first attempts towards directing the focus of an end-to-end communication on content by providing retrieval mechanisms as overlays over the existing Internet infrastructure and by exploiting the availability of cheap storage and processing capabilities in the existing Internet infrastructure. This comes in contrast to the traditional network protocols that were developed by focusing on the transfer of information from a source to a destination as quickly as possible, without considering other uses of the content [7]. While these protocols are very efficient in delivering content from an original source, they are not able to shorten the delay from a source to a destination by making use of storage or processing power within the infrastructure.

CDNs have been conceived to serve application-layer content to clients, such as Web pages, from servers that are as close to the client's physical location as possible. The operation of CDNs is based on clusters of servers distributed through the Internet infrastructure and the replication of content from content providers to these clusters. The selection of servers that will host the replicas is the result of constant monitoring and load-balancing operations applied on a network's traffic, while taking into account the Quality of Service (QoS) guarantees that content providers require [8]. Based on the physical location of clients, requests are directed through the Domain Name Servers (DNS) of the CDN provider towards the closest servers that hold replicas of the requested content.

P2P networks are based on the exchange of content between end users; each end user acts as a client as well as a server. P2P networks are more open concerning the traffic they serve, including files, movies, songs, etc, but they are limited in the sense that they use specific protocols for their operation [8], [9]. In P2P networks, a directory server resolves content requests into a number of potential sources named *peers*. In contrast to the replicas made in CDNs, peers in P2P networks may not have the full object requested but only parts of it, i.e., *chunks*. These parts of a content resource may be defined by an application, e.g., parts 1 to 7 of a movie or by the P2P protocol. Content requests, thus, may be propagated to one or more peers simultaneously and depending

TABLE I
A COMPARISON OF CONTENT-CENTRIC TECHNOLOGIES SHOWS THE ASPECTS DERIVED FROM ICN NETWORKING AGAINST EXISTING TECHNOLOGIES SUCH AS CDNS AND P2P NETWORKS AND ITS DIFFERENCES TO THESE TECHNOLOGIES

Characteristics	**CDN**	**P2P**	**ICN**
Infrastructure Type	Distributed/Clustered	Distributed	Distributed
Protocol Type	Proprietary	Application-specific	Standardized
Layer in OSI Stack	Application	Application	Network
Naming Type	Source-related	Source-free	Source-free
Naming Granularity	Object	Chunk	Object/Chunk/Packet
Serving Population	Proprietary Infrastructure	End-users	Internet Infrastructure

on the number of peers, reduce the overall retrieval time of content.

Despite the advantages of CDNs and P2P networks over the traditional network protocols, their performance and acceptance is limited due to their operation at the application layer and the technological and commercial administration boundaries that they apply [8], [10]. As the original design of the Internet was conceived to support a basic set of services, future changes on its infrastructure should guarantee backwards compatibility. Therefore, CDN and P2P technologies have been developed as overlays over the current Internet infrastructure, resulting in isolated deployments that do not share a common naming or encoding of their data units, thus, embed the interaction and content sharing among each other.

Information-Centric Networking (ICN) provides an alternative to the traditional end-to-end communication model of the current Internet architecture by focussing on information dissemination and information retrieval. ICN aims to address the shortcomings of CDNs and P2P networking by defining a common protocol - similar to the network layer - that can be used by various application-layer solutions while exploiting the processing power and storage within the infrastructure to replicate content. In order to achieve this, ICN routers are equipped with cache memory and enabled with a caching capability. Table I provides a comparison of the characteristics of CDNs and P2P networking against the ICN technology.

Approaches to caching in ICN may be categorized into: i.) *off-path caching* and ii.) *on-path caching* with regard to the location of caches [11], [12]. Off-path caching aims to replicate content within a network regardless of the delivery path(s), from a source to a consumer(s). On-path caching is conducted along a delivery path(s). On-path caching is integrated with the forwarding mechanism of ICN networking.

The contribution of this paper lies in the area of ICN caching and on-path caching in particular by providing a survey of the existing caching policies and forwarding mechanisms that complement these policies. In contrast to a number of ICN networking surveys [11]–[13] that focus on providing an overview of ICN architectures and energy-efficient solutions, our work constitutes a continuation and an expansion of a former survey [14] on caching policies and forwarding mechanisms by focusing on a detailed description of their operation through the use of examples and a description of their performance evaluation. A number of criteria, such as the caching model and caching level of operation, are then being employed to provide a taxonomy of the existing caching policies and a categorization according to the caching criteria used. Based on the information extracted by both the taxonomy and the categorization of the existing caching policies, one can observe: i.) the evolution of the caching criteria and caching policies, ii.) the effect of such criteria on the performance of a network, iii.) the caching policies and the parameters that have been used as benchmarks. Finally, a discussion that is based around the observations made in this paper is used to highlight the current trends and evaluation issues of caching policies in ICN networking, perform a qualitative comparison with regard to the advantages and disadvantages of the existing caching policies and suggest directions for future work.

The remainder of this paper is structured as follows. Section II provides an overview of the ICN networking communication model. Section III is dedicated to the description of the caching problem in ICN, highlighting the differences between on-path and off-path caching. Section IV contains background information such as categorization and terminology. Section V presents a summary of the on-path caching challenges. Section VI is dedicated to the identification of forwarding mechanisms for the exploitation of replicas. Section VII provides a detailed analysis of the existing caching policies, including their benefits, limitations and evaluation results against the alternatives. Section VIII highlights the current trends and evaluation issues in on-path caching, the advantages and disadvantages of the existing caching policies and the necessity for further research. Section IX closes this paper by presenting the conclusions derived from this survey.

II. INFORMATION-CENTRIC NETWORKING: AN OVERVIEW

The purpose of this section is threefold. Firstly, a brief description of the main features in ICN networking: i.) the publish-subscribe paradigm and ii.) the naming scheme are given; a description of the caching feature is given in Section III. Secondly, a discussion on the applicability of ICN networking on the current Internet infrastructure is being maintained. Thirdly, for ease of understanding, a description of the CCN-ICN networking communication model [2] is being employed. A complete overview of the features of ICN networking may be found in the following surveys [11], [12].

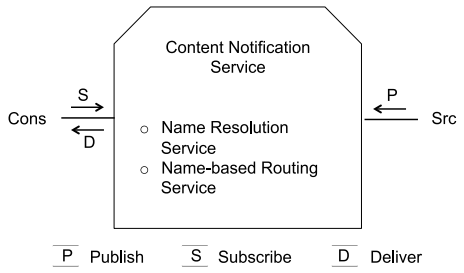


Fig. 1. Publish-Subscribe paradigm.

A. Publish-Subscribe Paradigm

ICN architectures, such as CCN [2], COMET [15], PSIRP-PURSUIT [16] and NetInf-SAIL [17], are based on the publish-subscribe paradigm [18]. In a publish-subscribe paradigm, content sources make their content available by publishing to a *content notification service* and consumers request content by subscribing to a content notification service [19], [20]. A content notification service may consists of: a *name resolution service* and/or a *name-based routing service* [11], [12]. An example of the aforementioned operation is presented in Fig. 1, where abbreviation *Cons* is used to define a consumer while abbreviation *Src* is used to define a source/publisher. For the rest of this paper and unless otherwise stated, abbreviations *Cons* and *Src* will be used to illustrate the operation of both the existing caching policies and the forwarding mechanisms. The application of the publish-subscribe paradigm ensures time, space and synchronization decoupling between publishers and subscribers [2] by moving away from a traditional connection-based approach and relying on the underlying infrastructure to mediate between publishers and subscribers.

B. Naming Scheme

The common medium that enables the interaction between the sources, consumers and components of the notification services is a standardized naming scheme. Similar to Internet Protocol (IP) addresses, a naming scheme allows the components of a publish-subscribe solution to interact without the necessity of tight coupling between them. Thus, naming becomes a key feature of ICN networking, able to affect the scalability and security of various mechanisms, such as caching and routing, and enable functionalities such as authentication and replication. In the design of content namespaces, a number of decisions need to be made such as: i.) naming may reveal structural information vs. naming does not reveal structural information, ii.) human-readable naming vs. not human-readable naming, iii.) flat naming vs. hierarchical naming, and iv.) self-certifying naming vs. not self-certifying naming [11], [12].

ICN naming schemes identify content resources such as services, Web pages, files, videos, etc, or parts of a content resource, chunks or packets using a *content identifier*. While chunks describe parts of a content resource as defined by an application, packets indicate a division of a content resource as defined by the network architecture. The type of content to which a name refers defines the naming granularity of the architecture, i.e., *object, chunk* or *packet naming granularity*.

Naming in ICN is considered to be unique and application-agnostic. Content identifiers should not involve information that would bind the content to a specific location [1], [21], [22]. Furthermore, all identical contents should share the same content identifier. If these constraints are met, content can be freely replicated and cached in different locations within a network, therefore, enabling functionalities such as in-network caching, multi-homing and multi-path forwarding.

C. Applicability of ICN

The exact benefits and limitations of ICN networking against the existing content-centric technologies are yet to be defined [23]. To this end, Arianfar *et al.* [24] and Perino and Varvello [25] provided a qualitative analysis of a CCN router; a router that supports the CCN-ICN networking communication model [2] (Section II-D). In the former, the authors are focused on the functionality and efficiency of a CCN router with regard to caching. Their analysis indicates that according to the existing memory technologies, the cache size of a CCN router is limited to 10Gb. In the latter, the authors consider a wider range of mechanisms such as caching, named-based routing and name-based forwarding with regard to both software and hardware. Their analysis indicates that both aspects need to be evolved to support a global deployment of the CCN architecture, while the architecture may be currently supported in a CDN or an Internet Service Provider (ISP)-scale network. Their conclusion is based on the scalability concerns with regard to the routing state that is required in order to support an Internet-scale name-based routing mechanism [26].

Towards exploring the benefits of on-path caching, Saino *et al.* [27] and Dräxler and Karl [28] provided a quantitative analysis of on-path caching in comparison to off-path caching. According to the results, on-path caching suggests lower performance gains than its alternative. Hence, on-path caching should not be considered as an alternative to off-path caching and should be rather used complementarily. To enchance the performance of on-path caching, Thomas and Xylomenos [29] proposed a cache structure, the *Object-oriented Packet Cache (OPC)*, that reduces the number of indexes by caching sequencial packets of objects, instead of standalone packets. This way, OPC is expected to reduce the lookup time of caches. However, the feasibility of OPC at this point of time is rather questionable, since a complex algorithm is required to allocate memory to objects, while memory requirements are practically unknown. Moreover, OPC is based on the use of sequence numbers which reflect the sequence of packets with regard to the first packet of an object and are part of the packets' format. This assumption may be invalid in cases where a multi-path forwarding mechanism is being used or no sequence numbers exist. The evaluation of OPC is subject to future work.

In addition to the aforementioned qualitative and quantitative analyses, *Network Coding (NC)* has been proposed as a technology able to enchance the performance of ICN

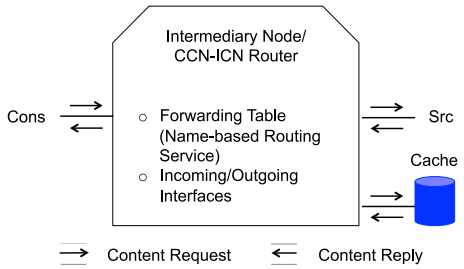


Fig. 2. An abstract representation of the CCN-ICN networking communication model.

networking by providing an alternative content dissemination technique [30]. NC is based on the application of algebraic equations on bits of packets [31]. Therefore, NC is able to linearly combine individual packets. According to experiments carried out on the current Internet infrastructure, NC has been shown to increase the resilience to content losses, while decreasing bandwidth consumption via the use of multi-path and multi-homing mechanisms [32]. Since, both multi-path and multi-homing mechanisms are natively supported in ICN networking, similar benefits are also being expected. In contrast to the aforementioned advantages, NC is expected to increase the complexity of both consumer and source nodes due to the use of algebraic equations and the necessity that rises to distinguish the individual packets at their final destination. The definition of a tradeoff between the benefits and limitations of NC technology on ICN networking is subject to future work.

D. ICN Networking Communication Model: An Example

In an ICN networking communication model, the term *content request* defines a request of content and the term *content reply* defines the delivery of content. Content requests are issued by consumers. Content replies are issued by any node that has the content, such as a content source/publisher or a router. Both, content requests and content replies contain content identifiers which specify the content that they serve.

As an example, Fig. 2 provides an abstract representation of the CCN-ICN networking communication model [2]. Based on the use of a name-based routing service, CCN is one of the most mature ICN networking architectures. The CCN-ICN networking communication model may be defined as follows: A content request is issued by a consumer. Content requests are propagated based on the forwarding table of a node. As content may be available by multiple sources, forwarding tables may hold multiple entries, i.e., faces, for a content identifier. At this point, it is worth noting that a face in the CCN-ICN networking communication model is an extended version of an interface as defined in the current Internet infrastructure in the sense that a face may also represent the communication between the CCN-ICN networking communication model and an application process within the same node, i.e., publisher/consumer. In order to be able to deliver content replies to consumers, a node keeps track of a list that consists of the incoming faces where content requests have been received. Upon the arrival of a content request, a node searches its cache. If the content is found, it is propagated to the consumer using the information maintained in the tracking list; consequently, content requests and content replies are forwarded via the same forwarding path(s), a feature called *symmetric forwarding*. If the content is not found, the content request is propagated towards a source based on the information of the forwarding table. Upon the propagation of a content request, a node keeps track of the outgoing face(s) used. This information is then used to verify the validity of content replies and discard unsolicited content.

CCN nodes have been also enabled with the feature of *request aggregation*. According to this feature, content requests that refer to the same content are aggregated and only the first of them is forwarded towards a source, thus, reducing the network traffic. Request aggregation is based on the assumption that content propagation is accomplished using content identifiers. For the same reason, nodes are also enabled with a caching feature that allows them to cache content replies. This feature introduces the problem of on-path caching which is explained in the following subsection.

III. Caching in ICN

Caching has been suggested as a complementary architectural feature in ICN networking, able to improve the Quality-of-Service (QoS) experienced by consumers, reduce the overall network traffic, prevent network congestion and Denial-of-Service (DoS) attacks and increase content availability.

Off-path caching, also referred to as *content replication* or *content storing*, aims to replicate content within a network in order to increase content availability regardless of the use of delivery paths. The actual number of replicas and the selection of nodes in which replicas may be stored is determined by monitoring and traffic analysis operations [8], [33]. This follows from the fact that storage placement decisions may be affected by contextual information, such as node availability, storage availability and content popularity [34].

The balance of this large set of variables results in a complicated system, sensitive to short-term changes which have the potential to significantly increase the complexity of a replication mechanism. In addition to this, replicas in off-path caching are usually advertised in a name resolution service and/or a name-based routing service [11], [12]. The distribution of replicas and the update of content-notification services that reference these replicas introduce additional traffic into a network that needs to be taken into account when deciding on the placement of replicas. Consequently, off-path caching should be considered as a long-term decision.

Due to the aforemenetioned prerequisities, content replication usually refers to the replication of content at an object naming granularity and is performed using complementary servers, located at the edge-points of a network [11]. Section V-B justifies this choice by providing information with regard to the cache size of intermediary nodes, i.e., routers. According to this information, the maximum cache size of a router is estimated to be 32Gb [35], which is relatively small in order to support content replication operations. In addition to this, both monitoring and traffic analysis operations limit

TABLE II
A COMPARISON OF OFF-PATH AND ON-PATH CACHING TECHNOLOGIES IN ICN NETWORKING

Characteristics	**Off-path caching**	**On-path caching**
Purpose	Increase Object Availability	Leverage Network Traffic
Relation to Forwarding Mechanism	None	Strict
A Priori Knowledge/Monitoring	Sufficient	None/Minimal
Content Naming Granularity	Object	Chunk/Packet
Decision Validity	Long-term, e.g. days/months/years	Short-term, e.g. seconds/hours/days
Caching Location	Edge-nodes, i.e. servers	Intermediary-nodes, i.e. routers
Content Replica Advertisement	Applicable	None/Minimal
Content Decision Technique	Optimization	Heuristics
Caching Decision Boundaries	Limited, i.e. ISP/AS	None

the scale where content replication may be applied to the size of an ISP or an Autonomous System (AS).

Given that the optimal placement of content replicas in a distributed storage system has been proven to be a hard task to achieve [36], [37], off-path caching is usually performed using optimization techniques [38], [39]. As a result, the problem of off-path caching in ICN is comparable to the CDN content replication and Web cache placement problems [11], [12]. Due to this similarity, only a small number of off-path caching mechanisms have been proposed, such as [27], [28], and [40]–[42], complementing the existing literature on optimized distributed caching [43]–[46].

In contrast to the existing literature on off-path caching and optimized distributed caching, on-path caching adds to this the integration of naming and caching mechanisms into the network layer of ICN networking. In more detail, on-path caching aims to reduce the traffic and the delay experienced in a network rather than increase content availability.

On-path caching is performed at the routers of a network, following either a chunk or a packet naming granularity. Therefore, on-path caching is limited by the use of forwarding paths and the cache sizes that ICN routers are bounded to follow (Section V-B). In addition to this, on-path caching is highly affected by the dynamic enviroment of ICN networking, that constitutes operations such as monitoring, collection of statistical information and advertisement of the cached content into a content notification service impractical and inefficient [47]. Due to these limitations, on-path caching is considered to be a short-term decision where the use of ligh-weight and heuristic tecnhiques is preferable to the use of optimization techniques. An important advantage of this is the applicability of on-path caching decisions to a wider scale rather than the boundaries of an ISP or AS. Table II provides a comparison of the characteristics of off-path caching against on-path caching in ICN networking. A detailed description of the challenges in on-path caching is presented in Section V.

IV. BACKGROUND: CATEGORIZATION AND TERMINOLOGY

The purpose of this section is to familiarize the reader with the categories and terminology used in this paper. In more detail, a description of the performance metrics and network topologies used in the evaluation of the existing caching policies is being provided. A categorization of the existing caching policies based on the caching criteria used, the caching model and caching level of operation are also being employed to distinguish the caching policies between one another.

A. Evaluation Metrics

Due to the integration of caching into the network layer of ICN networking, performance metrics may be categorized into: i.) *network-based metrics* and ii.) *cache-based metrics*.

1) Network-Based Metrics: Network-based metrics are used to measure the effectiveness of caching policies by measuring the traffic rates in a network, the distribution of the network traffic among the nodes in a network and the QoS experienced by consumers. Network-based metrics are calculated per forwarding path or per network-scale such as an ISP or an AS.

a) Hop-count: The *hop-count* metric represents the number of nodes being traversed by a content request before it is satisfied. To determine the traffic reduction achieved within a network due to caching, the hop-count metric is defined as a fraction, called *hop-count ratio*, *hop-count rate* or *hop-reduction ratio*, normalized by the hop-count that a content request would have traversed if no caching had been conducted. The hop-count metric has been shown to satisfactorily estimate the delay metric in a network [48], [49]. An increase of this metric corresponds to an increase of the network's traffic.

b) Download time: The *download time* metric indicates the delay experienced by consumers since the expression of a content request and the arrival of the corresponding content. Delay reduction and traffic reduction constitute the primary objectives of in-network caching. The download time is also referred as *latency*, *delay* or *Round-Trip Time (RTT)*.

c) Load fairness: The *load fairness* metric is defined as the fraction of content requests and content replies received by a node to the total number of content requests and content replies expressed in a network. This metric is used to indicate the existence of overloaded nodes.

d) Link traffic: The *link traffic* metric represents the amount of traffic on a link. Similarly to load fairness, the link

traffic metric indicates whether the traffic is being distributed within a network or concentrated to specific parts of it.

e) Network traffic: The *network traffic* metric is defined as the sum of link traffic values experienced within a network. This metric is commonly used to determine the traffic that exits an ISP, hence, the reliance of an ISP to its peers or parent tiers.

f) Responses per request: The *responses per request* metric corresponds to the number of content replies being received for a content request. This metric provides an estimation of the network traffic introduced by the multi-path mechanism.

2) Cache-Based Metrics: Cache-based metrics are used to measure the effectivess of caching policies by measuring whether a policy is able to cache and maintain the desired content. Cache-based metrics are usually calculated per node.

a) Server hits: The *server hits* metric refers to the number of content requests satisfied by a server, thus, indicating the load of a server. To determine the load savings of a server due to caching, the server hits metric is defined as a fraction, called *server-hit ratio*, normalized by the number of content requests expressed in a network. Preferable caching policies should conclude on a reduction of this metric. The server-hit ratio is also referred as *server-hit rate* or *server-hit reduction*.

b) Cache hits: The *cache hits* metric refers to the number of content requests satisfied by the cache of a node, different to the source node. To determine the load savings of a server due to caching, the cache hits metric is defined as a fraction, called *cache-hit ratio*, normalized by the number of content requests expressed in a network. Server-hit ratio and cache-hit ratio are controversial metrics. Preferable caching policies should correspond to an increase of this metric. The cache-hit ratio is also referred as *cache-hit rate* or *cache-hit probability*.

c) Cache evictions: The *cache evictions* metric refers to the number of cache replacements that occur on a node, using a cache replacement policy. An increase of this metric suggests lower exploitation of the cached content. This metric is also referred as *cache-replacements rate* and *cache-evictions rate*.

d) Caching efficiency: The *caching efficiency* metric is defined as the number of content requests satisfied by a cache and is calculated as the fraction of the cache hits on a node to the number of contents stored in the cache of the same node.

e) Caching frequency: The *caching frequency* metric is defined as the number of contents being cached on a node to the number of contents traversed through the same node.

f) Cache diversity: The *cache diversity* metric refers to the number of distinct contents stored in a network. This metric defines a tradeoff between the metric of content redundancy and the performance of caching policies.

g) Cache redundancy: The *cache redundancy* metric refers to the number of identical contents stored in a network. Cache diversity and cache redundancy metrics complement one another. Preferable caching policies should conclude to an increase of the former and a decrease of the latter metric.

h) Absorption time: The *absorption time* metric is defined as the time for which content remains cached in a node's cache. Absorption times depend on the cache replacement policy and the contents' popularity distribution. This metric is also referred as *caching time*.

B. Network Topologies

On-path caching policies have been evaluated using a range of network topologies. In this section, a description of these network topologies and a discussion of their characteristis and suitability for the evaluation of caching policies is being given. A representation of the topologies can be found in Fig. 3.

1) String Topology: String topologies are loop-free topologies where a single communication path is maintained for each pair of nodes in a network. In a string topology scenario, consumers are usually placed at one end of the topology and sources are placed at the other. String topologies are unable to recover in cases where a link failure or node failure occurs.

String topologies contradict the distributed nature of ICN networking and multi-path forwarding mechanisms. Therefore, string topologies are considered to be improper for the evaluation of caching policies in ICN networking. An example of a string topology is presented in Fig. 3(a).

2) Diamond Topology: Diamond topologies constitute an extension of string topologies. Similar to string topologies, consumers are placed at one end of the topology and sources are placed at the other. The two ends of the topology are then connected through a number of parallel communication paths that define the *level* of the diamond topology.

Due to the definition of multiple parallel communication paths, diamond topologies may be used to explore the effect of multi-path forwarding mechanisms on the performance of caching policies. However, diamond topologies do not constitute a representative example of Internet network topologies. Thus, the conclusions derived should not be generalized. An example of a diamond topology is presented in Fig. 3(b).

3) Grid Topology: Grid topologies are also an extension of string topologies where a string topology is stacked on top of another. In a grid topology, nodes are connected both horizontally and vertically. Grid topologies maintain multiple communication paths between a pair of nodes in a network.

Due to their consistent connection degree, grid topologies may be used to explore the performance of caching policies related to graph-based metrics (Section VII-B3). However, similar to diamond topologies, grid topologies do not constitute representative examples of Internet network topologies. An example of a grid topology is presented in Fig. 3(c).

4) Binary-Tree Topology: Binary-tree topologies constitute a subclass of k-ary tree topologies. K-ary trees are structured in layers, the number of which defines the *depth* or *level* of the tree. Parameter k defines the *degree* of the tree which is the number of nodes that a node is connected at its lower layer. A binary-tree is then a k-ary tree of $k=2$. In a binary-tree topology, a content source is usually placed at the root of the tree while consumers are placed at the leafs of the tree.

Thus, binary-tree topologies have been based on the assumption of hierarchical network structures. Even though, this assumption may be valid if the corresponding network-scale is limited, its validity tends to be erroneous as the network-scale increases. As a result, binary-tree topologies do not constitute representative examples of Internet network topologies. Hence, the conclusions derived should

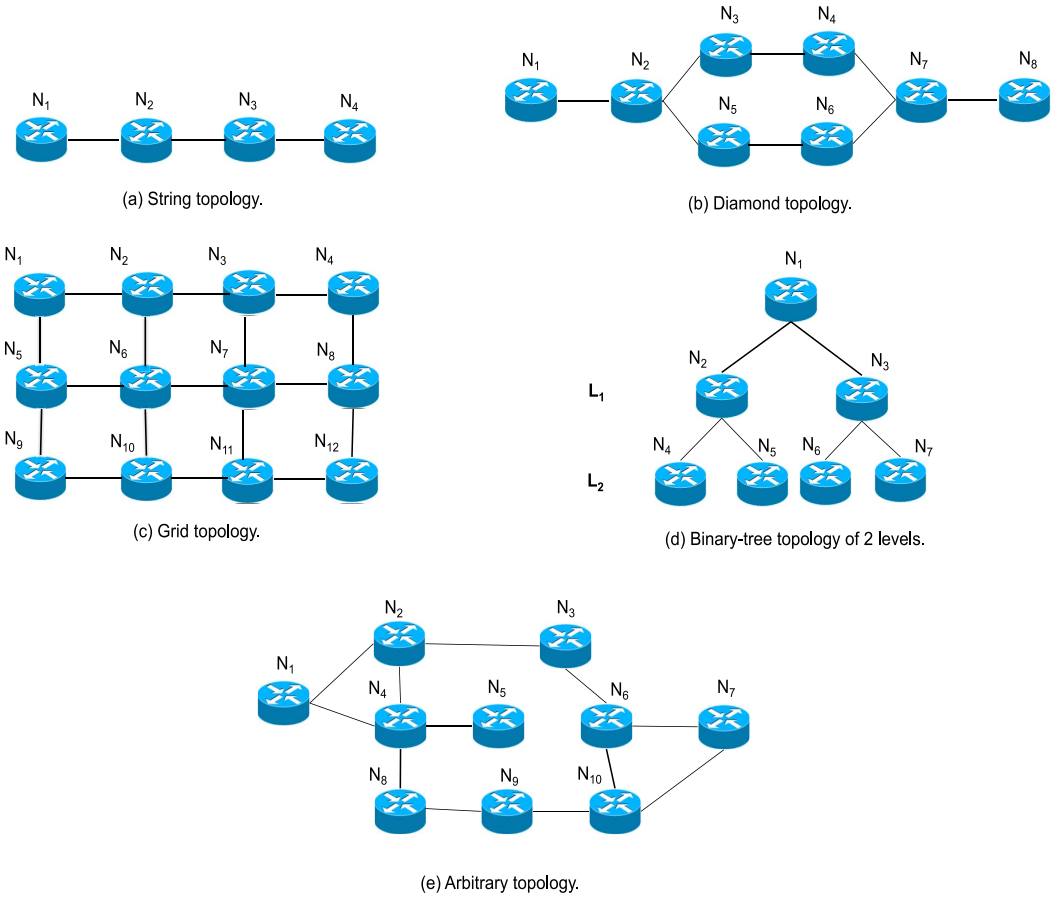


Fig. 3. Representation of the network topologies used in the evaluation of the existing caching policies.

not be generalized. An example of a binary-tree topology of N levels is presented in Fig. 3(d).

5) Arbitrary Topology: In contrast to the aforementioned network topologies, ICN networking has been defined to operate on arbitrary network topologies such as scale-free topologies and topologies extracted from the Internet infrastructure; for the rest of this paper the term *Internet-like topologies* will be used to refer to the latter category of network topologies.

Examples of Internet-like topologies are the Rocketfuel topologies [50], CAIDA topologies [51] and the topologies generated using the GT-ITM simulation tool [52]. An example of an arbitrary topology is presented in Fig. 3(e).

C. Categorization of Caching Policies

On-path caching is a complementary feature of ICN networking, able to, among others, alleviate the scalability and performance issues caused by large catalog sizes (Section V-D). Caching policies aim to distribute copies of content on the delivery paths to reduce content delivery times while limiting the number of replicas. In this section, a categorization of the existing caching policies with regard to a number of criteria such as the information used to perform a caching decision or the type of information being cached is provided.

1) Caching Criteria: Caching policies may be categorized based on the caching criteria used: i.) graph-based criteria and topological-based criteria such as the position of a node within a network or the distance of a node from a source, ii.) content-based criteria such as content identifiers or content popularity, iii.) a combination of criteria joined using a mathematical formula to construct a probability. Based on the aforementioned criteria, caching policies may be categorized as: i.) *graph-based caching policies*, ii.) *label-based caching policies*, iii.) *popularity-based caching policies* and iv.) *probabilistic caching policies*, respectively.

2) Caching Model: Caching policies may be categorized depending on the nature of information used for the caching decision. Based on this criterion, caching policies may be categorized as: i.) *autonomous caching policies*, ii.) *centralized caching policies* or iii.) *cooperative caching policies*, either *implicitly* or *explicitly*, if they use local information, centralized information or information with regard to a group of nodes in a implicit or explicit manner, respectively.

3) Caching Level: Following the discussion on naming granularity (Section II-B), caching policies may be categorized depending on the level of information they cache. Based on this criterion, caching policies that operate on a packet, chunk or object naming granularity may be categorized as: i.) *packet-based caching policies*, ii.) *chunk-based caching policies* and iii.) *object-based caching policies*, respectively.

V. On-Path Caching Challenges

The purpose of this section is to provide a description of the challenges in ICN networking with regard to on-path caching.

These challenges derive mainly from the following features: i.) the naming scheme, according to which content is defined using unique content identifiers, iii.) the caching mechanism, according to which routers are equipped with cache memory and enabled with a caching capability and iii.) the forwarding mechanism that determines from where information is forwarded, an essential feature of ICN networking that may be used for the exploitation of replicas. For the rest of this paper, the terms node and router are used interchangeably.

A. Caching Decision Applied on Specific Content, Nodes and Time

In contrast to existing caching technologies such as CDNs and Web proxies, where caching decisions may be determined a priori [8], a caching decision in on-path caching can only be applied on content that has been requested. A caching decision that does not introduce additional traffic within a network can only be applied on nodes that lie on the delivery paths. In order to support the caching of contents at alternative locations, a number of prerequisities need to be fulfilled, among which, the creation of additional replicas and mechanisms that would propagate these replicas to the required locations. Examples of such caching mechanisms have been shown to introduce more traffic and complexity within a network [53]–[55].

Since on-path caching takes place on the delivery paths and on content that has been requested, the time that a caching decision requires to be accomplished is limited to the time that content is propagated through an intermediary node. This time adds to the content delivery time experienced by consumers. Therefore, the time required for caching decisions to be accomplished should be limited and the occurrence of timeouts and request retransmissions should be prevented.

B. Limited Cache Capacity

Similar to the time required for a caching decision, the time required for the lookup of content in a cache is also being added to the content delivery time. In order to limit the lookup time, the cache size of a node has to be bounded. The recommended cache size lies in the range [10,32]Gb [24], [35]. These sizes have been concluded based on the assumption that caching is performed on a packet naming granularity.

C. Low Overhead Replacement Policies

Limited cache sizes require the use of content replacement policies. The requirement for low content delivery times affects not only the caching policies but the replacement policies as well, highlighting the need for low overhead implementations. Based on this criterion, content replacement policies such as the Least-Frequently Used (LFU) and Least-Recently Used (LRU) have been defined to be more complex to operate on an ICN node while shown to provide minor benefits against the RaNDom replacement policy (RND). The advantages of LRU and LFU against RND have been established through experimental results [24], [56], [57]. However, no numerical results have been presented to define the complexity overhead of the replacement policies while LRU has been adopted in a considerable number of caching policies, e.g., [58]–[60].

At this point, it is worth noting that content replacement policies and content placement policies may complement each other. To this end, a number of content replacement policies have been proposed with regard to on-path caching [61], [62]. Al-Turjman *et al.* [61] proposed a content replacement policy which is based on a utility function that takes into account the delay for retrieving content as well as the popularity and age of content experienced by a node, named *Least-Value First (LVF)*. The impact of a parameter in this utility function may be adjusted using weights. Wang and Bensaou [62] suggested a range of replacement policies depending on the location of a node in a network. If this node is an edge-node, i.e., a node connected to consumers, a replacement decision is performed using an alteration of the GreedyDual-size algorithm [36], based on the number of hops to fetch the content. If this node is an intermediary node, a replacement decision is performed using an alteration of the GreedyDual-size algorithm based on both the number of hops to fetch the content and the number of faces where a request for this content has been received.

D. Large Catalog Sizes

The term *catalog size* describes the number of individual contents in a network. Since content identification is usually performed on an object naming granularity and according to the amount of information available in today's Internet infrastructure, the catalog size with regard to ICN networking is estimated at 10^{12} [23] objects at minimum. However, content in ICN networking may also be defined in chunks or packets. A chunk naming granularity or a packet naming granularity results in an increase of the number of contents handled by a content notification service, raising both deployability and scalability concerns [63], [64]. Deployability and scalability concerns may, as well, affect the performance and efficiency of caching policies. In order to examine this possibility, large catalog sizes should be considered for their evaluation. For the rest of this paper and unless otherwise stated, the term catalog size will refer to the number of objects in a network.

E. Caching Cooperation

Cooperative caching has been suggested as a solution to increase content diversity in a network by exchanging availability information between nodes. Nodes exchange information with regard to the content they possess, which in turn allows them to eliminate content redundancy. Cooperative caching can be applied either globally or locally, i.e., *global-cooperative caching* and *local-cooperative caching*, respectively. In the former, the cooperation is carried out between all nodes in a network while in the latter, the cooperation is carried out between a group of nodes in a network.

Global cooperation is applied in cases where optimization techniques are used for the replication of content. Global-cooperative caching has been applied on content-centric technologies such as off-path caching [40]–[42], CDNs [48], [65] and Web proxies [66]–[68]. Similar to off-path caching and

content replication mechanisms, global-cooperative caching is a long-term decision that involves a number of prerequisities, such as monitoring, exchange of statistical information and advertisement of the cached content in a content notification service. This communication overhead and complexity, in addition to the dynamic environment of on-path caching, constitute the application of global-cooperation disadvantageous for short-term decisions. Consequently, only local-cooperative caching mechanisms may be applied in ICN.

According to the existing literature, two types of local-cooperative caching have been defined: i.) *path-based cooperative caching* and ii.) *neighborhood-based cooperative caching*. In the former, the cooperation is carried out between the nodes on a delivery path. In the latter, the cooperation is carried out between the neighbors of a node [53]–[55].

F. Arbitrary Network Topologies

On-path caching decisions can be applied on any node within a network that is equipped with caching capabilities. As a result, on-path caching is not limited to a specific network topology and can be applied to either *scale-free topologies* [69]–[71] or topologies derived from the Internet infrastructure [50], [51], Internet-like topologies (Section IV-B5); in scale-free topologies the degree distribution of nodes follows a power law [72]. Evaluations results on caching policies have highlighted the similarity of scale-free topologies and Internet-like topologies [73].

G. Mixed Population Patterns

According to a number of analyses on network traffic traces derived from the current Internet infrastructure, content popularity may be characterized by a Zipf distribution [74], [75]. This claim comes in contrast to popularity patterns defined in content-centric technologies such as P2P networks that have been shown to follow a Zipf-Mandelbrot distribution [76]. As Internet traffic tends to be more content-oriented, the use of a Zipf-Mandelbrot distribution is more suitable for the evaluation of caching policies in future Internet architectures.

However, according to recent measurements on BitTorrent's traces [77], Zipf laws may not completely characterize the distribution of content popularity. The distribution of content popularity appears to be composed by three parts: the head, the waist and the tail. Summarizing the results, content popularity distribution is affected by the sampling time interval as follows: Zipf laws may correctly characterize content popularity distributions over short time intervals such as 1 week while the popularity of contents over long time intervals such as 4 weeks to 48 weeks, does not exhibit a power-law tail but an exponential cutoff. Since content popularity distributions define which content should be cached, this dependency may significantly affect the performance of on-path caching policies.

More recently, Imbrenda *et al.* [78] analyzed the traffic of an ISP network using both HypeText Transfer Protocol (HTTP) requests and replies. According to their findings, content popularity may, indeed, be characterized by more than one power law. Content popularity has been shown to follow a Weibull distribution for the head and tail of the distribution and a Zipf distribution for the waist.

H. Correlated Content Requests

Existing caching policies [59], [62], [79] have been based on the assumption of independent content requests, following the *Independent Reference Model (IRM)*. The IRM model has succesfully served in-network caching mechanisms such as CDNs and Web proxies [80]–[82], where an object naming granularity is assumed. However, content requests in ICN networking may correspond to the retrieval of pieces of a content resource, resulting in a chunk or a packet naming granularity, thus, being correlated to one another. Consequently, the assumption of IRM arrivals may not always be valid.

Correlated arrivals have been examined using a *Markov Modulated Rate Process (MMRP)* [83], [84]. According to this model, objects are divided into K categories with each category corresponding to a different averaged *Virtual Round Trip Time (VRTT)*. Content requests are issued following a chunk naming granularity. VRTT is the time between the transmission of a content request and its satisfaction. Objects in class $k \epsilon K$ are requested using a Poisson distribution of $\lambda = \lambda_k$ while a class is chosen using a uniform distribution. Content requests are issued sequentially, i.e., upon the satisfaction of a content request, a subsequent content request is issued.

Dabirmoghaddam *et al.* [85] have questioned the validity of the IRM model with regard to object naming granularity. According to this proposal, content requests often exhibit both *spacial locality* and *time locality*. Spatial locality is based on the fact that content requests that refer to the same object are more likely to be issued by geographically close areas. Time locality is based on the fact that subsequent content requests are more likely to refer to objects that have been recently requested. According to further analysis on network traffic traces that highlight the occurance of both locality types [86], [87], Dabirmoghaddam *et al.* proposed a reference model that considers both locality aspects as well, defined by a temporal locality factor, named *localization factor (l)*, where $l \epsilon [0, 1]$. A system model with $l=1$ corresponds to an IRM model.

I. Exploitation of Cached Content

The exploitation of replicas is the primary objective of in-network caching and on-path caching. An essential component of ICN networking that affects the performance of caching policies is the forwarding mechanism, which determines from where information is forwarded. Based on the criterion whether a forwarding mechanism is aware of the replicas made using a caching policy, forwarding mechanisms may be categorized into: i.) *opportunistic forwarding mechanisms* and ii.) *caching-aware forwarding mechanisms*, respectively.

In the former, no information with regard to the replicas of contents is maintained. A replica is exploited if the node that possesses the replica lies on the forwarding path of a subsequent content request. Opportunistic forwarding has been adopted as the default forwarding strategy by the majority

of caching policies, e.g., [2], [16], [24], [59], [63], where a Shortest-Path Routing (SPR) mechanism is assumed.

In the latter, forwarding is based on the collection of information with regard to the replicas of contents. Hence, caching-aware forwarding is more likely to benefit the performance of a caching policy. In its simplest form, this collection of information may be accomplished by advertising the replicas into a content notification service. However, the advertisement of replicas, in addition to the challenges and restrictions that on-path caching applies, e.g., large catalog sizes (Section V-D) and low overhead operations (Section V-C), constitute the use of content notification services at the network layer of ICN networking inefficient, resulting in out-of-date information and increased network traffic [63], [64]. As a result, alternative methods for the collection of information with regard to the available replicas within a network should be considered. To this direction, a number of caching-aware forwarding mechanisms have been proposed. A description of these mechanisms is provided in the following section (Section VI).

VI. Caching-Aware Forwarding

The purpose of this section is to provide a review of the caching-aware forwarding mechanisms in ICN networking. Due to the strict connection between on-path caching policies and forwarding mechanisms [88], a number of caching-aware forwarding mechanisms have been based on the functionality of the caching policy being used. Such forwarding mechanisms are explained in conjunction with the operation of the corresponding caching policy while the rest of the forwarding mechanisms are explained individually.

A. Random-Based Forwarding

Domingues *et al.* [89] proposed a hierarchical ICN architecture that is based on a forwarding mechanism called *Random-based Forwarding (RFW)*. RFW is based on random walks. The architecture consists of a logically hierarchical network topology, where consumers are placed at the lowest layer, layer N, and publishers are placed at the highest layer, layer 1.

Upon the transmission of a content request at layer N, the content request is forwarded in layer N following a random walk for a maximum time T; a random walk is defined as the random choice of an outgoing face upon the arrival of a content request on a node. If a content request has not been satisfied by the time T, it will be forwarded to the next higher layer in the architecture. In order to achieve this, all nodes that belong to layer $n \epsilon N$ should be connected to a subset of nodes of the same layer as well as nodes that belong to the layer above, i.e., gateway nodes that belong to layer $n-1$. The process of forwarding a request within a layer and subsequently forwarding it to the next layer will be repeated until the request is either satisfied by a cache or reaches layer 1. At layer 1, the request will be satisfied by the publisher.

Fig. 4 presents a hierarchical network of N=3 layers, where publishers are placed at layer 1 and consumers are placed at layer 3. Time T is represented by the number of hops that a content request is eligible to traverse and is equal to T=2. Upon the propagation of a content request, the value of T is decreased by one. When T=0, the content request is forwarded to the next higher layer of the architecture. Following the aforementioned assumptions, a content request takes a random walk from node N_{11} to node N_{14}. Upon reaching node N_{14}, T is decreased by one, hence, T=1. Since, the requested content has not been found, yet, and $T \neq 0$, the content request is forwarded further, at this time to node N_{15}. Once again, T is decreased by one, hence, T=0. Since, T=0 the content request is forwarded to the next higher layer of the architecture, layer 2. This operation is illustrated by the dashed arrow.

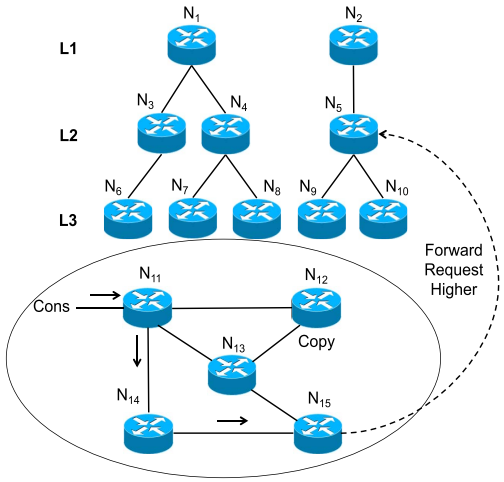


Fig. 4. Random-based forwarding is based on randomly selected forwarding paths on each layer of a logically hierarchical network topology.

During the forwarding of a content request, a trail of breadcrumbs is installed on each node traversed. The trail consists of: i.) the content identifier, *inf*, ii.) a *back pointer* that points to the previous node traversed and iii.) a *reinforced counter*, *rc(inf)*, which is used to determine whether to cache the corresponding content. A *rc(inf)* counter has two thresholds: i.) the upper threshold, *rc-up(inf)* and ii.) the lower threshold, *rc-low(inf)*. Upon the arrival of the requested content on a node and if *rc(inf)* is equal to *rc-up(inf)*, the content is cached in the node and the back pointer is deleted. In the case that *rc(inf)* is equal to *rc-low(inf)* and if the content is cached in a node, the content is immediately removed. To be able to reach either thresholds, *rc(inf)* is increased upon the arrival of a content request and decreased periodically by rate $\gamma(inf)$. The parameters *rc-up(inf)*, *rc-low(inf)* and $\gamma(inf)$ are set based on the assumption that content demand remains fixed and is known a priori. For the rest of this paper, we will refer to this caching policy as the *Reinforced-Counters Caching* policy.

Due to the operation of RFW on a hierarchical architecture, a tradeoff is defined between time T and the delay experienced by consumers as well as the traffic experienced by layer 1. If a content is more likely to be available on a specific layer, an increase of parameter T is being suggested. If otherwise, a decrease of the same parameter is being suggested.

B. Learning-Based Forwarding

Learning-based forwarding mechanisms are composed of two phases: i.) the *exploitation phase* and ii.) the *exploration* or

discovery phase. In the former, a node is using the forwarding path(s) in its forwarding table. In the latter, a node attempts to discover replicas by exploring alternative paths. A forwarding decision is then based on both the discovered paths and the already known paths. This is because the discovery of a new forwarding path does not guarantee the discovery of a replica.

1) INFORM: INFORM [90] is a dynamic forwarding mechanism based on the Q-learning algorithm [91]. INFORM provides a formal representation of its processor, a forwarding mechanism proposed by Chiocchetti *et al.* [92]. In INFORM, a discovery phase is accomplished through flooding. To prevent content requests flooding a network, flooding is based on an exponentially fading probability and performed for popular contents only. In the case of a replica discovery on a path, a decision on which path is preferable is made based on the RTT. Even though the selection of a forwarding path is based on the RTT metric, other metrics may be used instead.

2) Stateful Forwarding: Stateful forwarding [93] is another forwarding mechanism based on learning. A rank value that states the preference of a forwarding path over an alternative path is used to make forwarding decisions. The determination of a rank value depends on a number of performance metrics such as the RTT, timeout rates and link status, according to which a link may be ranked to be either *green, yellow* or *red*. A link is ranked green when no delivery failures or congestion have been acknowledged, yellow when congestion has been acknowledged and red when transmission is unlikely to occur, e.g., a link failure has occured. Upon the arrival of a content request on a node and if the content is locally unavailable, green paths will be preferred to yellow paths while red paths will be excluded from the forwarding procedure.

Using specific paths to forward content requests, increases the likelihood of generating bottlenecks in a network and decreasing the overall performance. Depending on the configuration, stateful forwarding may propagate periodical content requests, *probing messages*, through random paths to discover existing replicas. This phase corresponds to the discovery phase defined in the former subsection. During the discovery phase, both green and yellow paths may be explored. Periodic probing may be accomplished upon the expiration of a time interval or when a certain number of content replies have been received on an face. Periodic probing limits the traffic introduced in a network due to the discovery mechanism.

Fig. 5 illustrates the operation of a network using a learning-based forwarding mechanism. For the purpose of this example, a forwarding decision is made based on the RTT metric alone, while all links experience the same delay. Hence, the RTT metric of a forwarding path is equal to the number of hops traversed. Consumer *Cons* is linked to source *Src* through a number of forwarding paths: i.) $(N_1, N_2, N_3, N_6, N_{10})$, ii.) $(N_1, N_4, N_5, N_6, N_{10})$ and iii.) $(N_1, N_7, N_8, N_9, N_{10})$. Since, all paths correspond to the same RTT, $(N_1, N_2, N_3, N_6, N_{10})$ has been randomly chosen to be the primary forwarding path. At time t, an alternative forwarding path, i.e., $(N_1, N_7, N_8, N_9, N_{10})$, is explored by sending a content request through this path. The content request will be satisfied by node N_8 that possesses a copy of the requested content. Upon the arrival of a content reply at node N_1, N_1 will compare the RTT of both the primary path and the explored path, and select the one with the lowest value; $(N_1, N_7, N_8, N_9, N_{10})$ is selected to be the new primary path, as its RTT is less than the current one, i.e., 2 versus 4. At this point, it is important to highlight that the preference to a forwarding path may be based on either the existence of multiple sources or content copies. Consequently, the validity of this preference may correspond to long time-intervals or short time-intervals, respectively.

C. Flooding-Based Forwarding

Flooding-based forwarding mechanisms have been proposed as a solution for the exploration of replicas within a network and the reduction of traffic on the default forwarding paths, such as the shortest routing paths. Their operation can be explained as follows: upon the arrival of a content request on face i of a node, the content request is forwarded through all faces besides i, i.e., $\forall j \epsilon I, j \neq i$, where I is the set of faces. In the following, a description of the existing flooding-based forwarding mechanisms, with regard to the exploration and the exploitation of replicas, is being given.

1) Ideally-Nearest Replica Routing (iNRR): Recent proposals on on-path caching have defined the replica exploitation problem as an optimization problem [94], according to which, the exploration phase refers to the discovery of all replicas while the exploitation phase refers to the use of the nearest replicas. *ideally-Nearest Replica Routing (iNRR)* is based on the assumption that complete knowledge with regard to all replicas is being provided. Due to the fact that the aforementioned assumption is practically inefficient, two alternative forwarding mechanisms have been proposed that approximate the performance of iNRR: i.) *ideal-Nearest Replica Routing' (iNRR')* and ii.) *ideal-Nearest Replica Routing" (iNRR")* [88].

iNRR' and iNRR" are based on the adaptation of flooding techniques. iNRR' extends the operation of learning-based forwarding, (Section VI-B), on mutliple paths while iNRR" differs from iNRR' as follows: upon the arrival of a content request on a node and if the content is locally available, an empty content reply is sent instead of a content reply. Empty content replies correspond to data-free payloads. The idea behind empty content replies is that data-free payloads will prevent the occurrence of cache evictions and the increase of network traffic rates that would have been caused by flooding content replies. Since content replies contain data, more bandwidth is required for their transmission while their arrival on a node triggers the execution of a caching policy.

In addition to the difference between iNRR' and iNRR", both mechanisms differ from the rest of the forwarding mechanisms in a number of ways: i.) a temporary forwarding table is assumed for their operation, ii.) a *scope-value*, similar to a Time-To-Live (TTL) value and iii.) a hop-counter that is being updated during the forwarding process, are injected in the content request's packet format. Upon the arrival of a content request and if the content is locally unavailable, a node will forward the request according to its temporary forwarding table. If a matched entry does not exist, a node will forward the request according to its forwarding table. A scope-value is

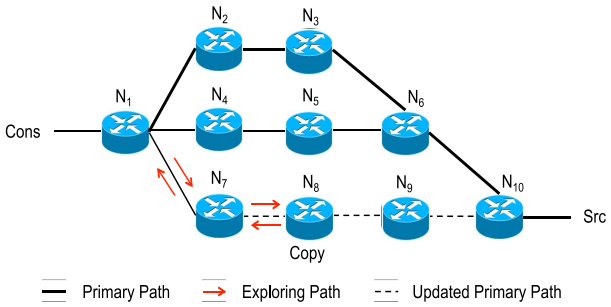


Fig. 5. Learning-based forwarding is based on the exploration of a number of alternative forwarding paths in a network.

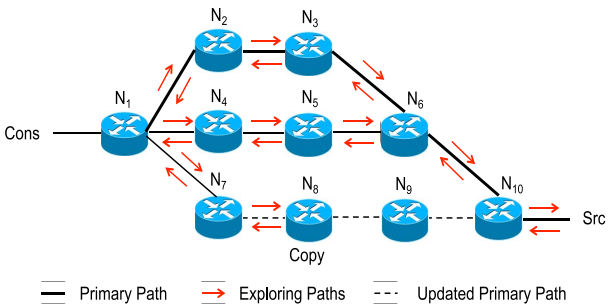


Fig. 6. Flooding-based forwarding is based on the exploration of all forwarding paths that cross over a single node in a network.

used to prevent infinite flooding of content requests in a network. This category of flooding is named *scope-flooding*. In order to achieve this, each time that a content request reaches a node, its scope-value is reduced by one. If positive, the content request is flooded and discarded otherwise. A hop-counter is used to update the routing metric of forwarding faces in the temporary forwarding table each time that an exploration phase is completed, i.e., a content reply has been received. To this end, upon the arrival of a content request on a node, the hop-counter is increased by one. This hop-counter is then injected in the content reply's packet format.

2) Congestion-Based Search (SEARCH-CNG): In contrast to the scope-flooding forwarding mechanisms that are based on the hop-count metric, Badov *et al.* [95] proposed a scope-flooding forwarding mechanism that is based on a link-weight parameter, named *Congestion-based Search (SEARCH-CNG)*. A link-weight parameter serves the same purpose as a scope-value of a content request. Considering an example where the link-weight parameter is the reverse bandwidth of a link, a content request will be flooded until a link with as low bandwidth as the one that has been defined is found. The bandwidth of a link is defined as the fraction of the link's capacity to the number of flows over this link. A forwarding decision is then made based on the minimum bandwidth of the links between the node that has sent the content request and the nodes that have sent a content reply. To obtain this value, a field that holds the minimum bandwidth value is injected in both the content request's and content reply's packet format.

The goal of SEARCH-CNG is the identification of paths with low delay rates and is closely related to the functionality of the *Congestion-Aware Caching (CAC)* policy used. CAC aims to decrease the download times experienced by consumers. In order to achieve this, CAC exploits the available cache capacity to avoid the use of congested links by caching content on the downstream end of a congested link. CAC is executed on each node on a delivery path and is based on two factors: i.) the download time experienced by a node, defined as the fraction of content size to the minimum bandwidth value and ii.) the content popularity, defined as the fraction of the number of requests received on a node for a content to the total number of requests. Depending on the calculation of the minimum bandwidth value, two modifications of the CAC policy have been defined: i.) *CAC-E2E* and ii.) *CAC-UP*. In the former, the minimum bandwidth value is calculated based on both the upstream and the downstream bandwidth. In the latter, this calculation is based on the upstream bandwidth alone.

Fig. 6 illustrates the operation of a scope-flooding forwarding mechanism. The scope-value of flooding is set to be equal to the hop-count metric between the consumer and the source, i.e., *scope-value=4*. Upon the arrival of a content request at node N_1, N_1 will forward the request through all its functional faces, besides the face where the content request was arrived, i.e., nodes N_2, N_4 and N_7. Upon the arrival of a content request at node N_7, the content request will be satisfied by its cache while the rest of the content requests will be satisfied by the source. For the purpose of this example, it is assumed that the selection of a forwarding path as a primary path is based on the RTT metric while all links experience the same delay. Following this assumption, the RTT metric of a forwarding path will be equal to the number of hops traversed. Consequently, N_7 will sent a faster content reply than the source and N_1 will change its primary path to $(N_1, N_7, N_8, N_9, N_{10})$.

D. Index-Based Forwarding

Index-based forwarding mechanisms are based on the maintenance of indexes in nodes that reference the location of replicas, either permanent or temporary [90], [92], [96], [97]. Index-based forwarding mechanisms involve a number of decisions such as the specification of nodes that are eligible to hold indexes, the index format and whether nodes are eligible to perform caching or not. Based on these decisions, a range of forwarding mechanisms may be defined. Depending on the application scale, i.e., path or neighbors, index-based forwarding mechanisms may be categorized into: i.) *path-indexed forwarding mechanisms* and ii.) *neighborhood-indexed forwarding mechanisms*, respectively. A disadvantage of index-based forwarding mechanisms is consistency. Inconsistent indexes correspond to incorrect redirections of content requests that increase both the network traffic rates and content delivery times. In addition to this, index-based forwarding is based on the exploitation of topological information, a feature originated from the end-to-end communication model [97].

1) Potential-Based Routing (PBR): Potential-Based Routing (PBR) [96] is the routing protocol of the *Cache Aware Target Identification (CATT)* architecture, an ICN architecture that considers the integration of routing and caching mechanisms while ensuring availability and robustness.

In order to achieve this, CATT is designed as a distributed architecture constructed by the edge nodes of a network, the *Cache Aware Target Nodes (CATNs)*. CATNs execute the PBR protocol and are responsible for the maintainance of routing information with regard to publishers, permanent replicas and temporary replicas made using an on-path caching policy.

While publishers are assumed to advertise the content they possess to their local CATN directly, permanent and temporary replicas are advertised to CATNs via update messages, named *advertisement packets*. Advertisement packets are forwarded in a network using scope-flooding. Upon the arrival of an advertisement packet on a CATN node n, an index that defines the *potential field* (ψ_n) of a content is being created; ψ_n is defined as the fraction of the content's quality at node n_c that possesses the replica to the distance between n_c and n. The calculation considers all the existing replicas of a content, either permanent or temporary, combined using a summation formula. Quality and distance parameters may be interpreted into a range of variables such as the processing power of a node or the hop-count and the transmission delay.

Upon the arrival of a content request on a CATN node and if the content is locally unavailable, a node will search its indexes constructed by the PBR protocol and forward the request to the CATN node that possesses the content. If otherwise, the content request is forwarded towards a publisher. CATNs update their indexes on their own decision to ensure consistency. PBR is neither a path-index nor a neighborhood-index forwarding mechanism but a global-indexed forwarding mechanism leveraged by the metric of scope-value.

2) Scalable Content-Aware Networking Routing (SCAN): A neighborhood-indexed forwarding mechanism has been proposed by both Wang *et al.* [55] and Lee *et al.* [98]. Even though, the aforementioned proposals do differ from each other, they both share the same indexing mechanism. According to this mechanism, one-hop neighbor nodes periodically exchange summaries of their cached contents via bloom filters. These summaries are then kept as indexes. Each time a content request arrives on a node, the node checks its cached contents to locate the one requested. If the content exists, it is forwarded towards the requestor. If the content does not exist, the node searches its indexes to define whether the content is available at one of its neighbors. If none neighbor has the content, the node propagates the content request to a publisher.

3) Content-Oriented Network With Indexed Caching (CONIC): Content-Oriented Network with Indexed Caching (CONIC) [97] is an ICN architecture that aims to exploit the available storage within a network and eliminate the redundant traffic by maintaining in-network indexes. CONIC differs from CATT in the following aspects: i.) not only the edge nodes, but any node within a network is eligible to hold indexes, ii.) replicas are stored at the consumers rather than the intermediary nodes of the network's infrastructure. Nodes on the delivery paths decide whether to reference a replica using a probability p. This probability is calculated based on the fraction of the distance of a node from the node that possesses the replica to the sum of this distance and the distance to the consumer that has issued the content request, measured in hop counts. Upon the arrival of a content request, a node decides to forward the content request further or redirect it to a consumer according to its indexes information.

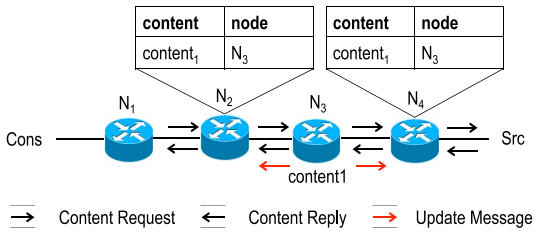


Fig. 7. Index-based forwarding is based on the maintainance of indexes that reference the location of replicas in a network.

Fig. 7 illustrates the operation of a neighborhood-indexed forwarding mechanism. For the purpose of this example, it is assumed that upon the arrival of an update message at a node from one of its neighbors, the node will always create an entry that references the content to its neighbor. Assuming that a consumer has transmitted a content request for $content_1$ and upon the arrival of $content_1$ at N_3, N_3 chooses to cache the content before forwarding it further to the consumer. In addition to its content reply message, N_3 will also propagate an update message towards its neighbors to inform them about its decision to cache the content. Both nodes N_2 and N_4 will then create an entry that references $content_1$ to N_3.

VII. On-Path Caching Policies

In this section, a detailed desrciption of the existing caching policies presented in this paper is given. The description consists of the operation of a caching policy and the evaluation results against the alternatives. In cases where the alternative caching policies have not been introduced yet, the evaluation results are provided in the subsections of the alternative caching policies. As an example, *Random-Probabilistic Caching (RND)*, (Section VII-A1a), is the first caching policy that is introduced, hence, no evaluation results have been provided. For ease of understanding, the operation of a caching policy is also being illustrated through the use of examples. A discussion with regard to the advantages and disadvantages of a caching policy is finally being employed.

Table V and Table VI provide a summary of the evaluation scenarios used for the evaluation of a caching policy against the alternatives. Table VI constitutes a continuation of Table V. To ensure consistency between the tables, the first two columns of Table V are duplicated to Table VI. Table V provides information such as the approaches against which a caching policy has been compared, the caching model and caching level of operation and the traffic model characteristics. Table VI provides information such as the evaluation metrics and network topologies. Despite the fact that tautological terms of the same metrics may be used to describe the evaluation of the existing caching policies, such as the hop-count ratio and hop-reduction ratio (Section IV-A1a), all metrics are unified in Table V. In both tables, symbol *"–"* is used to indicate that no further information has been provided for a given category. At this point, it worth noting that UniCache, a caching policy explained in Section VII-A1b

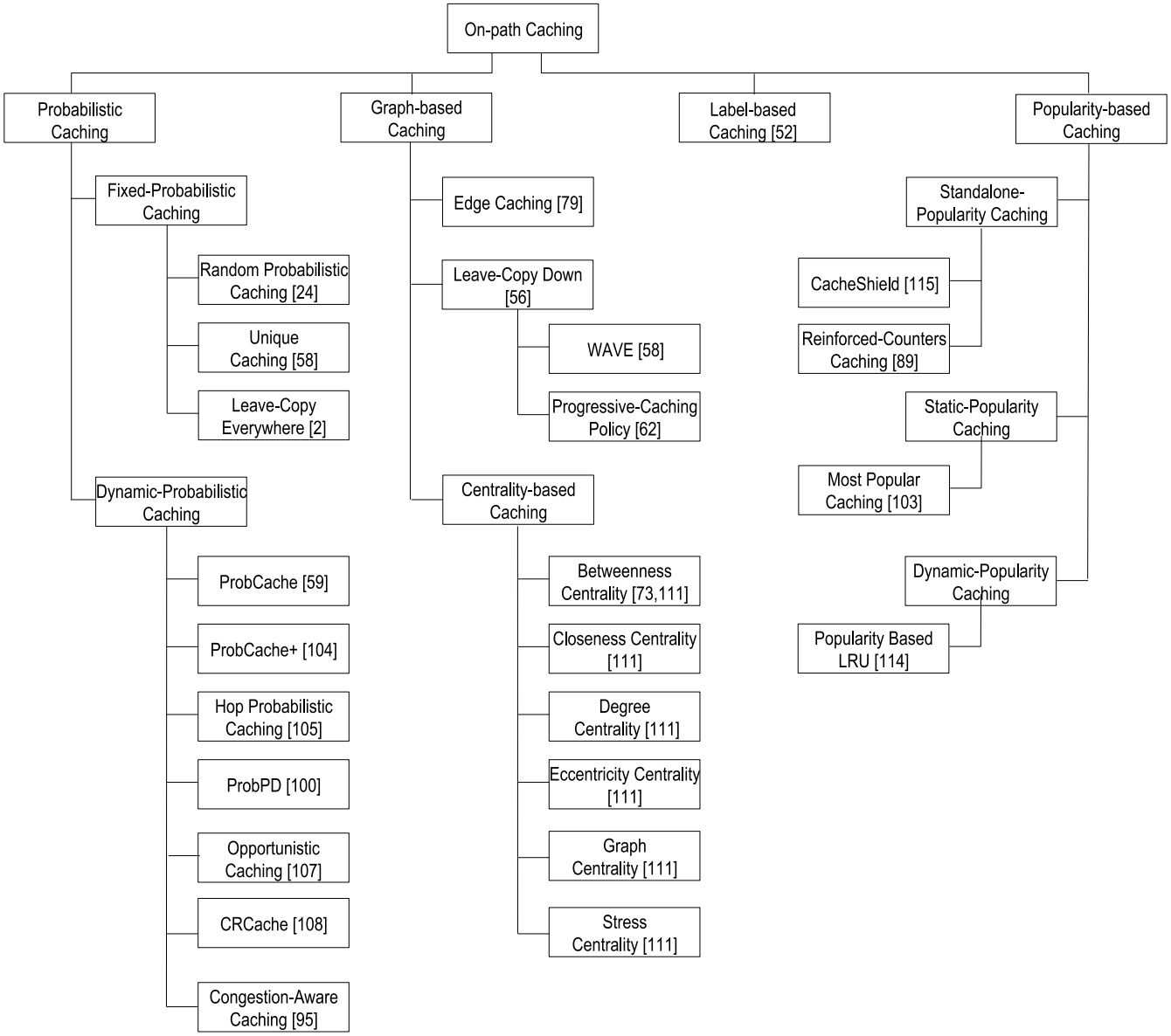


Fig. 8. A taxonomy of the existing on-path caching policies.

is only listed as a comparison policy as it has not been individually proposed. Based on the information of either Table V or VI, the evaluation of this policy may be found in Sections VII-A1c, VII-A2a, VII-A2d, VII-B2, and VII-B2a where the alternative caching policies are being explained. The evaluation results correspond to those concluded by the relevant literature.

Fig. 8 presents a taxonomy of the existing caching policies according to the caching criteria used. As an example, *Fixed-Probabilistic Caching* (Section VII-A1) is a subcategory of *Probabilistic Caching* (Section VII-A), where a caching decision is based on a fixed, a priori determined probability.

A. Probabilistic Caching

Probabilistic caching policies base their decision to create a replica on a node on a probability p [24], [99]. Depending on the structure of this probability, whether it is a fixed value or a mathematical formula consists of a number of individual components, probabilistic caching may be categorized into: i.) *Fixed-Probabilistic Caching (FIXp)* and ii.) *Dynamic-Probabilistic Caching*, respectively.

1) Fixed-Probabilistic Caching (FIX(p)): Fixed-probabilistic caching policies are based on a fixed probability defined a priori. Caching decisions are individually taken and involve no cooperation between the nodes of a network, therefore, do not increase the network traffic. For the same reason, fixed caching policies are unable to exploit any knowledge with regard to the content or the network topology. Consequently, fixed caching policies make no distinction on what content to cache or where to cache it.

a) Random-probabilistic caching (RND(p)): Random-probabilistic caching (RND(p)) is a form of fixed-probabilistic caching, where the probability p may be decided randomly, as long as $p \epsilon [0, 1]$. Random caching can be applied either globally or locally, i.e., i.) *globally-random caching* and

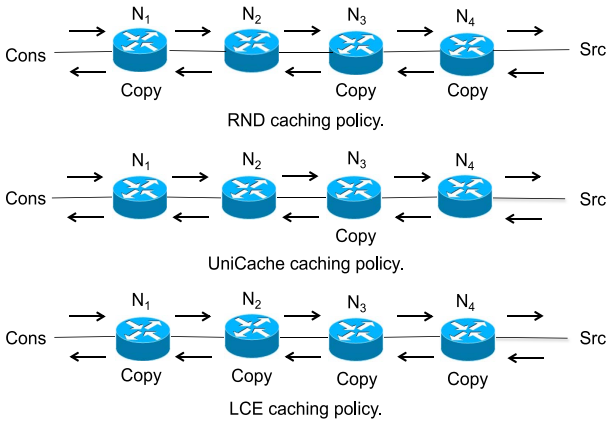


Fig. 9. Representation of fixed-probabilistic caching policies.

ii.) *locally-random caching*, respectively. In the former, all contents within a network share the same probability p. In the latter, the probability p is determined per content request, individually.

Even though, random caching policies share the advantages of simplicity and low overhead of fixed-probabilistic caching policies, their efficiency is rather questionable due to their unpredictable nature of randomness. As the factor of randomness is higher in locally-random caching policies, existing caching policies have only considered the application of globally-random caching [24], [56], [59]. Fig. 9 illustrates the operation of globally-random caching for p=*0.9* that causes nodes N_1, N_3 and N_4 to cache the content. Random caching policies have been commonly used as a benchmark in the evaluation of newer caching policies, concluding on an increase of several performance metrics such as server-hit rates, download times and cache redundancy against the alternatives [24], [59].

b) Unique caching (UniCache): Unique caching *(UniCache)* is a form of fixed-probabilistic caching where the probability p is uniformly distributed among the nodes of a delivery path, i.e., content is cached on only one node on a delivery path [58], [73]. In order to achieve this, the probability p is defined as the fraction of one to the number of nodes on a delivery path. Following the example in Fig. 9, the caching probability on each node is p=*1/4*. Consequently, low-stretch delivery paths will experience higher caching probabilities and high-stretch delivery paths will experience lower caching probabilities. This comes in contrast to the purpose of in-network caching to shorten content delivery times by caching content on the intermediary nodes of lengthy delivery paths while increases the possibility of caching redundant content on low-stretch delivery paths. Due to the aforementioned drawbacks, UniCache has been shown to increase both the server-hit ratio and the hop-count metrics [58], [73].

c) Leave-copy everywhere (LCE): Leave-Copy Everywhere (LCE) [2] is another form of fixed-probabilistic caching with p=*1* and the first caching policy that has been proposed in ICN networking. LCE caches a content on every node on a delivery path. An example of this policy is presented in Fig. 9. LCE has been used as a benchmark for the evaluation of a large number of newer caching policies, e.g., [53], [56], [59], resulting in high cache redundancy and resource consumption [99]–[101]. On the contrary, LCE provides the advantage of faster content dissemination than its alternatives [73], [102].

Evaluation [24]: LCE has been evaluated against a number of random caching policies under two evaluation scenarios: i.) *scenario I* and ii.) *scenario II*, using the metrics of cache-hit ratio and download times, respectively. Both scenarios have been based on an 8-node string topology for the retrieval of a single video. The caching operation is held on a packet level. The definition of both the object size and the cache size are being omitted. Scenario I consists of a single consumer that is placed at one end of the network topology and a source that is placed at the other. All caches are initially converged, i.e., the requested video has been previously downloaded and cached. Scenario II consists of 8 consumers that are placed at one end of the network topology and a source that is placed at the other. All caches are initially empty.

The results suggest an increase of 60% and 100% of the cache-hit ratio for RND(0.125) and RND(1), respectively, for scenario I, with the rest of the random caching policies, i.e., RND(0.5), RND(0.33) and RND(0.25) to be lying in between. A decrease of the download times has been recorded for scenario II, as the probability p increases, e.g., 819 seconds for RND(0.125), 689 seconds for RND(1.25) and 488 seconds for RND(0.5). The aforementioned results come in contrast to a number of alternative evaluations that suggest lower performance gains for LCE, with regard to the same evaluation metrics [59]. The reason for this contradiction is the variety of evaluation scenarios being used and the lack of the corresponding evaluation to consider the effect of a number of evaluation parameters, such as the cache size and the catalog size, on the performance of LCE. With that said, LCE will always outperform RND. However, such evaluation scenarios are not realistic. A discussion on the suitability of the evaluation scenarios is maintained in Section VIII-B.

2) Dynamic-Probabilistic Caching: In contrast to fixed caching policies (Section VII-A1), dynamic caching policies aim to react to the traffic patterns and the structure of a network by taking into account characteristics related to the content and the network topology. An example that calculates the dynamic probability of each dynamic caching policy is being illustrated through Sections VII-A2a–VII-A2f.

a) ProbCache: ProbCache [59] decides to cache content based on the number of replicas to be cached on a delivery path, determined by the cache capacity of a delivery path and a factor to counterbalance the tendency to cache solely far from a source(s). The goal of this policy is to provide fairness with regard to the capacity of a delivery path over different contents. In order to achieve this, contents which are destined to consumers that are further away from a source, should be less likely to be cached in nodes that are close to a source. The rationale behind this idea is that such contents will have more opportunities to be cached along a delivery path than contents that are destined to consumers closer to a source.

ProbCache, equation (1), is calculated on each node on a delivery path and is composed of two factors: i.) the *TimesIn*

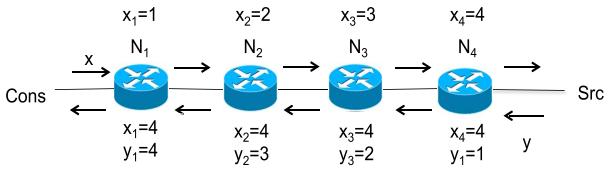


Fig. 10. Calculation of x and y values used in dynamic-probabilistic caching policies.

factor and ii.) the *CacheWeight* factor. The TimesIn factor, equation (2), favors contents that travel from further away while the CacheWeight factor, equation (3), acts as a counterbalance to this unfairness. The TimesIn factor depends on the variables: i.) N_n, ii.) T_{tw}, iii.) N_x, iv.) x and v.) y while the CacheWeight factor depends on the variables: i.) x and ii.) y.

N_n is the cache size of node n on a delivery path, represented by the time for which content should be cached in a cache. T_{tw} is the time for which content should be cached on a delivery path. N_x is the average cache size of a delivery path and is based on the cache size of the nodes on a delivery path. Given that this information is unkown, a node assumes that the cache size of the rest nodes on a delivery path is equal to its own. This assumption results in poor performance of the caching policy in heterogeneous cache networks, where the cache sizes differ between the nodes. Finally, x is the number of hops traversed from a consumer to a source and y is the number of hops traversed from a source to node n. During content delivery, the value of variable x remains stable while the value of variable y is being updated additively. The values of both variables are illustrated in Fig. 10. To make the figure as comprehensible as possible, arrows that show the direction of the calculations have been added. Based on the aforementioned variables, the TimesIn factor calculates the cache capacity of the remaining nodes on a delivery path while the CacheWeight factor is defined as the fraction of y to x. Due to the symmetric nature of forwarding mechanisms in ICN networking, (Section II-D), CacheWeight $\epsilon[0, 1]$.

$$ProbCache = TimesIn \times CacheWeight \tag{1}$$

$$TimesIn = \frac{\sum_{n=1}^{x-y+1} N_n}{T_{tw} \times N_x} \tag{2}$$

$$CacheWeight = \frac{y}{x} \tag{3}$$

To have an indication of the operation of ProbCache, Fig. 10 illustrates the values of the variables x and y on nodes: N_1, N_2, N_3, N_4; $x = (4, 4, 4, 4)$ and $y = (4, 3, 2, 1)$, respectively. For simplicity, all nodes are equipped with a cache size of $N_n = N_x = 10$ seconds. T_{tw} is set to be $T_{tw} = 2$ seconds. The caching probability on nodes: N_1, N_2, N_3, N_4 is equal to: $p = (0.5, 0.75, 0.75, 0.5)$, respectively. The aforementioned calculations are summarized in Fig. 11.

At this point, it is important to highlight a number of observations. Firstly, consulting Fig. 11, the nodes that are close to the end-points of a delivery path are shown to exhibit a lower caching probability while this probability increases symmetrically. In more detail, nodes N_2 and N_3 exhibit an equal caching probability, while this probability is higher than the

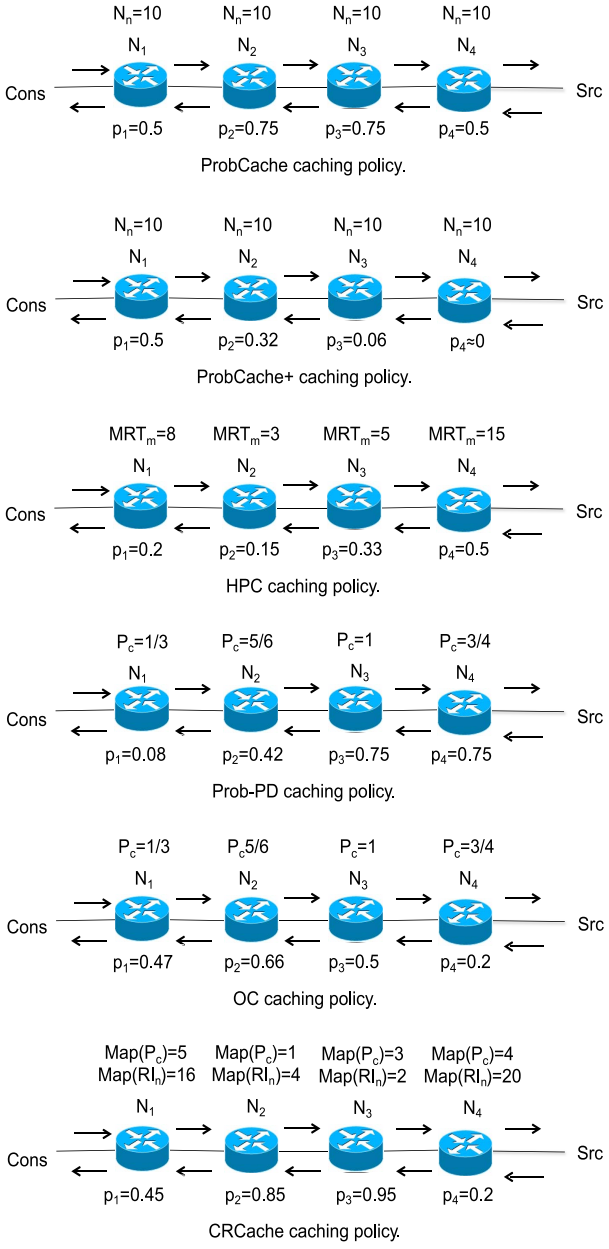


Fig. 11. Representation of dynamic-probabilistic caching policies.

one at nodes N_1 and N_4. This comes in contrast to the goal of ProbCache to provide fair allocation of the cache capacity on a delivery path among contents. Secondly, the operation of ProbCache implies that the values of variables x and y should be updated at each node on a delivery path. This prerequisite results in extra computational cost per node. This cost is related to the security aspects of ICN networking, such as authenticity and integrity. As an example, an authentication operation may be necessary to ensure that the packet has been modified by a valid node. Finally, ProbCache makes no distinction on what content to cache. Therefore, ProbCache fails to address the problem of cache pollution (Section VII-D).

Evaluation [59]: ProbCache has been evaluated against LCD, LCE, RND(0.3) and RND(0.7). The results indicate performance gains with regard to the metrics of server hits,

hop-count ratio and cache evictions. The evaluation has been based on a 6-level binary-tree topology with the source being placed at the root of the tree. Consumers are placed at the last two levels of the tree, generating a traffic model of 10^5 content requests that follow a Zipf distribution of $\alpha = 0.8$. At this point it is worth noting that neither the catalog size nor the number of consumers have been defined. The cache size N_n is set to be equal to $N_n \epsilon [1, 6]$ seconds. T_{tw} is set to be equal to $T_{tw} = 10$ seconds. Content is evicted from caches using a LRU replacement policy. According to Psaras *et al.*, both the N_n and T_{tw} parameters depend on the topology and the traffic patterns of a network. Therefore, both parameters should, only, be set by an ISP. The initialization of both parameters for the current evaluation scenario has been set arbitrarily.

ProbCache has been shown to provide a reduction of [12%,15%] of the metric of server hits against LCE, and a reduction of [7%,10%] against the remaining caching policies. The hop-count ratio metric of ProbCache has been calculated to be [8%,10%] higher compared to LCE and about [3%,4%] higher compared to LCD and RND(p). Depending on the level of the binary-tree topology, ProbCache has been shown to conclude to a reduction of [75%, 97%] of the cache evictions metric against the alternatives. The aforementioned results indicate that ProbCache is able to maintain the requested content longer than the alternative caching policies.

Nevertheless, the aforementioned evaluation lacks of a number of parameters that may considerably affect the performance of a caching policy. Examples of such parameters are: i.) the caching level of operation and ii.) the catalog size. Depending on the caching level of operation, memory requirements are expected to be higher for higher-level information, that follow an object naming granularity, while computational requirements are expected to be higher for lower-level information, that follow a packet naming granularity. The catalog size affects the likelihood of contents to gain a position in a cache.

b) ProbCache+: The unfairness of ProbCache has been verified for a range of T_{tw} values, $T_{tw} = \{5, 10, 15, 20\}$, using both experimental evaluation and theoretical analysis [103]. To enforce fairness between the contents, an enhanced version of ProbCache, *ProbCache+*, equation (4), has been proposed. ProbCache+ ensures that the caching probability is proportional to the distance of a node from a source. In order to achieve this, CacheWeight, equation (6), has been risen to the power of x. The behavior of ProbCache+ has been been shown to remain stable with regard to the same range of T_{tw} values.

$$ProbCache+ = TimesIn \times CacheWeight \tag{4}$$

$$TimesIn = \frac{\sum_{n=1}^{x-y+1} N_n}{T_{tw} \times N_x} \tag{5}$$

$$CacheWeight = \left(\frac{y}{x}\right)^x \tag{6}$$

To have an indication of the operation of ProbCache+, Fig. 11 illustrates the caching probabilities: $p = (0.5, 0.32, 0.05, 0)$, on nodes: N_1, N_2, N_3, N_4, respectively. For the purpose of this example, the caching probability on node N_4 is estimated to be: $p = 7.8 \times 10^{-3}$. Hence, a value of p=0 is used to indicate that the caching probability is negligible. Fig. 11 verifies that ProbCache+ corresponds to an increase of the caching probability as the content reply is being propagated towards a consumer. Nevertheless, ProbCache+ holds the remaining disadvantages deriving from its ancestor (Section VII-A2a). The calculations have been based on the same example as ProbCache, decsribed in Section VII-A2a.

Evaluation [103]: ProbCache+ has been evaluated against edge-caching, LCD, LCE and ProbCache, using the metrics of hop-reduction ratio, server hits, cache hits and cache evictions. According to Section VII-A2a, RND(0.3) and RND(0.7) yield similar performance gains to LCD. Hence, both caching policies have been excluded from the present evaluation.

The evaluation has been conducted using a scale-free topology of 200 nodes and 2 servers, that are placed at two nodes in the core of the network. Consumers issue a total number of 10^4 content requests, following a Zipf distribution of $\alpha = 1.2$. However, the distribution and the number of consumers is rather unknown. The cache size N_n is set to be equal to $N_n \epsilon [1, 6]$ seconds. T_{tw} is set to be equal to 10 seconds. The catalog size lies between the range of $[10^3, 10^4]$ of the cache size. Content is evicted from caches using a LRU replacement policy. Similar to ProbCache, ProbCache+ has been based on the same set of arbitrary values while the level of operation and the naming granularity of the system have not been defined.

ProbCache and ProbCache+ have been shown to outperform the alternatives by [4%,16%] and [10%,21%], respectively, with regard to the metric of server hits. ProbCache and ProbCache+ have been also evaluated against a range of T_{tw} values, $T_{tw} = \{1, 5, 10, 20\}$. The results suggest that low T_{tw} values, such as $T_{tw} = 1$, result in an increase of the metric of cache evictions, while high T_{tw} values, such as $T_{tw} = 20$, have no impact on the performance of the caching policies. The behavior of the caching policies has been shown to remain stable for a range of cache sizes, $N_n \epsilon [1, 6]$.

To examine the benefits of ProbCache and ProbCache+, with regard to the metrics of cache hits and cache evictions, a 6-node path of the network topology has been randomly extracted. According to the results, ProbCache corresponds to an increase of up to 80% of the cache hits metric and a decrease of [2.6%, 93%] of the cache evictions metric. ProbCache+ has been shown to outperfom the alternatives by a range of [7%, 88%] and [60%, 96%] of the same metrics. The aforementioned results verify the superiority of ProbCache+ against its ancestor, ProbCache. At this point it is worth noting that depending on the distance of a node from the source, both ProbCache and ProbCache+ may be outperformed by edge-caching, by a range of [60%, 85%], with regard to the metric of cache evictions. However, this behavior is considered to be an exception. Hence, a description of it is being omitted.

In addition to the aforementioned evaluation, a scenario where the content popularity is unexpectedly increased is also being considered. The goal of this scenario is to examine the convergence of the caching policies with regard to the factor of content popularity. To this end, an unpopular content is chosen, whose number of requests is set to be equal to 5% of the number of requests that are generated per second. In contrast to the majority of the caching policies, which correspond to similar performance gains, LCD and edge-caching conclude to an

increase of 20% of the metric of server hits and an increase of 10% of the metric of hop-reduction ratio. This result is derived from the fact that both ProbCache and ProbCache+ cache content at nodes that lie further away from the consumers, compared to LCD and edge-caching. LCE eliminates this effect due to its redundant nature of caching.

c) Hop-based probabilistic caching (HPC): The *Hop-based Probabilistic Caching (HPC)* [104], equation (7), is a probabilistic caching policy composed of two factors: i.) the $CacheWeight_y$ factor, equation (8) and ii.) the $CacheWeight_{MRT}$ factor, equation (9). The $CacheWeight_y$ factor aims to decrease content redundancy on a delivery path by progressively pushing content towards to consumers. The $CacheWeight_y$ factor is equal to the reverse distance between a node and the source that has satisfied the content request, y. The parameter y has been already defined in the description of ProbCache (Section VII-A2a). The distance is defined in hop counts and it is increased by a parameter α, where α is a constant integer determined according to the network's cache capacity. However, no further explanation of the calculation of parameter α is being given. For evaluation purposes, $\alpha = 1$. The $CacheWeight_{MRT}$ factor aims to store content for a specific time interval. In order to achieve this, the $CacheWeight_{MRT}$ factor is based on the variables of: i.) mean residence time and ii.) the expected mean residence time of contents on a node, MRT_m and MRT_{exp}, respectively. According to Wang *et al.*, [104] MRT_{exp} is defined a priori depending on the traffic load of a node. However, no further details on the determination of MRT_{exp} are given. The value of $CacheWeight_{MRT}$ is equal to the fraction of MRT_m to MRT_{exp}, if $MRT_m < MRT_{exp}$ and 1, if otherwise.

$$HPC = CacheWeight_y \times CacheWeight_{MRT} \quad (7)$$

$$CacheWeight_y = \frac{1}{y+\alpha}, \quad a \geq 0 \quad (8)$$

$$CacheWeight_{MRT} = \begin{cases} \frac{MRT_m}{MRT_{exp}}, & MRT_m < MRT_{exp} \\ 1, & MRT_m \geq MRT_{exp} \end{cases} \quad (9)$$

To have an indication of the operation of HPC, an example based on the values of parameter y, illustrated in Fig. 10, is being provided. For simplicity, all nodes correspond to $MRT_{exp} = 5$ seconds and $\alpha = 1$. According to Fig. 10, the values of the parameters y and MRT_m at nodes: N_1, N_2, N_3, N_4, are: $y = (4, 3, 2, 1)$ and $MRT_m = (8, 3, 5, 15)$ seconds, respectively. Hence, $CacheWeight_y = (0.2, 0.25, 0.3, 0.5)$ and $p = (0.47, 0.66, 0.5, 0.2)$. The aforementioned calculations are summarized in Fig. 11.

At this point, it is important to highlight a number of observations. Firstly, the value of the $CacheWeight_y$ factor decreases as the content reply is propagated towards to the consumer(s). This comes in contrast to the purpose of HPC to progressively push content towards to consumers. In more detail, the value of $CacheWeight_y$ is indistinguishable for contents that are originated from the same content source and despite the fact that they may be destined to different consumers. Secondly, the $CacheWeight_{MRT}$ factor depends on the mean value of the residence times. Mean values may yield to inaccurate estimations if the sample distribution is neither normal nor uniform. Thirdly, the $CacheWeight_{MRT}$ factor depends on a number of parameters defined a priori, such as the parameter α. To this end, HPC requires centralized information which comes in contrast to the distributed nature of ICN networking. Finally, similar to ProbCache, the factor of content popularity has not been taken into account.

Evaluation [104]: HPC has been evaluated against LCE and ProbCache using two network topologies: i.) a 7-nodes string topology and ii.) an arbitrary topology of 8 nodes. In the former, two consumers are placed at one end of the network topology and a source is placed at the other. In the latter, four consumers and three servers are arbitrarily placed within the network topology. Each server possesses a set of objects. The object size lies within $[1, 100]Kb$. Consumers issue object requests using a Zipf distribution of $\alpha = 0.85$. The caching operation is held on a chunk level. The chunk size is set to be equal to 1Kb. The cache size is set to be equal to 200 contents or chunks. Content is evicted from caches using a LRU replacement policy. In order to explore the benefits of HPC over different catalog sizes, two evaluation scenarios have been considered: i.) *scenario I* and ii.) *scenario II*. In the former, the catalog size is equal to 85 objects and equal to the caching capacity of the network, i.e., 14×10^2 and 16×10^2 chunks for the string and the arbitrary topology, respectively. In the latter, the catalog size is ten times the catalog size of scenario I, i.e., 850 objects and 14×10^3 and 16×10^3 chunks, respectively. The evaluation metrics include the cache-hit ratio, cache redundancy and absorption times.

HPC corresponds to an increase of 38% and 20% of the cache-hit ratio metric of LCE and ProbCache, respectively and a decrease of 25% of the cache redundancy metric of LCE. The cache redundancy rates of HPC against ProbCache have been shown to be equivalent. The aforementioned results have been based on scenario I for the string topology. An increase of 38% and 33% of the cache-hit ratio metric has been concluded against the same caching policies for scenario I for the arbitrary topology while no results have been provided with regard to the metric of cache redundancy.

Moving to scenario II, HPC suggests an increase of 12% and 9% of the cache-hit ratio metric of LCE and ProbCache, respectively, for the string network topology. HPC has been shown to be outperformed by LCE for the arbitrary network topology. In contrast to scenario I, no results have been provided with regard to the metric of cache redundancy for either network topology. To have an indication of the metric of absorption times, the average absorption times of node N_1 have been displayed for both evaluation scenarios for the string network topology. The results suggest an increase of [29%,34%] in favor of HPC against the alternatives.

At this point it is worth noting, that according to the aforementioned results, the performance of HPC is affected by both the catalog size and the network topology. However, no justification for the effect of such parameters on the performance of the caching policy has been given. Secondly, the superiority of HPC against the alternative caching policies is still an open question, since a consistent evaluation, using all the evaluation metrics, is missing. Thirdly, both evaluation scenarios have been based on unrealistic network topologies. A discussion

on the suitability of the evaluation scenarios is maintained in Section VIII-B while a brief discussion on the evaluation requirements of caching policies in ICN networking has been already defined in Sections V-D and V-F.

d) ProbPD: ProbPD, equation (10), is a probabilistic caching policy, composed of two factors: i.) the *Popularity* factor, equation (11), and ii.) the *Distance* factor, equation (12). *ProbPD* aims to address the shortcomings of caching policies, such as the ProbCache (Section VII-A2c), where caching unpopular content is likely to occur.

$$ProbPD = Popularity \times Distance \tag{10}$$

$$Popularity = \frac{R_c}{\sum_{\forall q \epsilon Q} R_q} \tag{11}$$

$$Distance = \frac{y}{x} \tag{12}$$

The idea behind the *Popularity* factor is that popular contents will satisfy a higher number of content requests. Thus, caching popular contents should be preferred and caching unpopular contents should be prevented. In order to achieve this, the *Popularity* factor is calculated per content and is equal to the number of content requests being received on a node, R_c, to the total number of content requests, $\sum_{\forall q \epsilon Q} R_q$, during a time interval ΔT. Thus, $Popularity \epsilon [0, 1]$. This type of content popularity is named *dynamic local-based popularity* and is explicitly described in Section VII-D3. To avoid introducing any overhead to the infrastructure of a node, ΔT is defined to be the time between the arrival of a content request and its satisfaction. The idea behind the *Distance* factor is that content should be placed closer to consumers to reduce the number of hops towards to a source. The *Distance* factor is equivalent to the CacheWeight factor of ProbCache (Section VII-A2a). Thus, ProbPD shares the same concerns about the calculation of variables x and y.

To have an indication of the operation of ProbPD, an example based on the values of the variables x and y, presented in Fig. 10, is being provided. For simplicity, the *Popularity* factor of content c is represented by the symbol P_c. The values of P_c at nodes: N_1, N_2, N_3, N_4, are arbitrary given and are equal to: $P_c = (1/3, 5/6, 8/8, 3/4)$, respectively. Consequently, the caching probability at nodes: N_1, N_2, N_3, N_4, will be equal to: $p = (0.08, 0.42, 0.75, 0.75)$, respectively. The aforementioned calculations are summarized in Fig. 11.

Evaluation [105]: ProbPD has been evaluated against DC, LCE, ProbCache and RND(0.9) using a 5-level binary-tree topology and the metrics of cache hits, cache evictions and download times; DC is a centrality-based metric that indicates the number of edges on a node and is described in Section VII-B3. A content soure of 10^3 contents is placed at the root of the tree. Single consumers are placed at the leafs of the tree. Consumers issue content requests using a Zipf distribution of $\alpha \epsilon \{1.0, 1.5\}$. The cache size is set to be equal to 10^2 contents. Cache evictions follow a LRU replacement policy. The naming granularity of the content unit is undefined.

ProbPD has been shown to outperform the alternatives by an increase of $[3\%, 99\%]$ and a decrease of $[83\%, 88\%]$ with regard to the metrics of cache hits and cache evictions for $\alpha = 1$, respectively. ProbPD has been also shown to perform closely to ProbCache with regard to the metric of cache evictions while DC has been shown to approximate ProbPD with regard to the metric of cache hits. The latter has been observed after the 30^{th} node of the binary-tree topology. In contrast to the aforementioned benefits, ProbPD results in an increase of 6% of the download time metric concluded for DC, LCE and RND(0.9). ProbCache results in an increase of $[4\%, 15\%]$ of the same evaluation metric for ProbPD.

Similar, ProbPD outperforms the alternatives by an increase of $[57\%, 97\%]$ of the metric of cache hits and a decrease of $[66\%, 90\%]$ of the metric of cache evictions for $\alpha = 1.5$. The difference of ProbPD with regard to the download time metric yields to an increase of 7% against DC, LCE and RND(0.9). The difference of ProbCache with regard to the same evaluation metric yields to an increase of $[11\%, 67\%]$ against ProbPD. The results indicate that the performance pattern of the caching policies shown for $\alpha = 1.5$ is equivalent to the performance pattern that has been observed for $\alpha = 1$.

The aforementioned results may be explained by the selective caching nature of ProbPD and the dependency of the *Popularity* factor on the feature of request aggregation (Section II-D), hence, the dependency of the *Popularity* factor on the network topology. With that said, the values of the *Popularity* factor with regard to the corresponding network topology tend to be higher towards the root of the tree since the number of content requests is more likely to be merged towards this point. This behavior is justified by both the increase of the cache hits and the download times metrics. The distinctive behavior of DC after the 30^{th} node can be explained as follows: since content is cached on nodes with the highest number of edges, the nodes $1^{st} - 30^{th}$ would be preferred over the nodes $31^{st} - 62^{nd}$. This means that the nodes $31^{st} - 62^{nd}$ will be less likely to cache content originated from the source. This also means that they will have more cache capacity to cache contents from the 3^{rd} level of the binary-tree, resulting in higher cache hits and lower cache evictions.

e) Opportunistic caching (OC): Opportunistic Caching (OC) [106], equation (13), is a probabilistic caching policy composed of two factors, which are equivalent to the factors of ProbPD (Section VII-A2d): i.) the *Popularity* factor, equation (14) and ii.) the *Distance* factor, equation (15). As a result, OC shares the same advantages to ProbPD.

$$OC = Popularity \times Distance \tag{13}$$

$$Popularity = R_c^{\beta} \tag{14}$$

$$Distance = \begin{cases} \frac{y}{x}, & cached = 0 \\ \frac{y - \alpha}{x - \alpha}, & cached \neq 0 \end{cases} \tag{15}$$

OC differs from ProbPD in the following aspects. Firstly, the *Popularity* factor of OC lies on the category of *standalone popularity* calculations (Section VII-D1) instead of the category of dynamic popularity calculations (Section VII-D3), while, no further information with regard to the calculation of this factor is given. According to the description of both the static and the dynamic popularity calculations, each category

experiences its own performance and implementation drawbacks. Hence, a conclusion on which approach is preferable would be rather invalid. Secondly, the Popularity factor has been risen to the power of $\beta\epsilon[0, 1]$, equation (14). The value of parameter β depends on the traffic patterns of a network and should be, onyl, set by an ISP. Thirdly, contents that have already been cached on a delivery path are less likely to be cached on the remaining nodes of a delivery path. The aforementioned decision is captured in equation (15), where α is the distance between a content source and the node that has cached the content. This way, OC aims to increase the content diversity on a delivery path. In contrast to the variables x and y, whose calculation has been explicitly defined in the description of ProbCache (Section VII-A2a), the calculation of variable α has not been defined. Last, OC differs from ProbPD in the sense that OC will always cache content if enough space exists.

To have an indication of the operation of OC, Fig. 11 illustrates the caching probabilities on nodes: N_1, N_2, N_3, N_4, i.e., $p = (0.47, 0.66, 0.5, 0.2)$, respectively. The calculations have been based on the values of variables x and y in Fig. 10. For simplicity, the *Popularity* factor of content c is represented by the symbol P_c. The values of P_c at nodes: N_1, N_2, N_3, N_4 are arbitrary given and are equal to: $P_c = (1/3, 5/6, 8/8, 3/4)$, respectively. The value of variable β is set to be equal to the benchmark value that is used in the simulations, thus, $\beta = 0.7$.

Evaluation [106]: OC has been evaluated against LCE, MPC, ProbCache, RND(0.3) and RND(0.9), under two evaluation scenarios: i.) *scenario I* and ii.) *scenario II*, using the metrics of hop-count, cache-hit ratio and cache eviction; MPC is a popularity-based caching policy and is described in Section VII-D2a. OC has been defined to operate in a packet caching level, following a packet naming granularity. Scenario I consists of a 3-level binary-tree topology and a traffic load based on synthetic traces. A content soure of 10^3 contents or packets is placed at the root of the tree. Single consumers are placed at the leafs of the tree. Consumers issue content requests using a Poisson distribution of $\lambda = 100$ and a Zipf distribution of $\alpha = 0.73$. For the purpose of the simulations, $\beta = 0.7$. At this point, it is worth noting that the value of variable β is set to be equivalent to the value of variable α. According to the results, the value of parameter β may affect the performance of OC. Scenario II consists of an Internet-like topology of 52 nodes and a traffic load based on real traces. A total number of 3 content servers, consists of approximately 7.1×10^3 contents are distributed randomly in the network. Each server stores an equal share of contents. The cache size is set to be equal to 10^2 contents. Contents in caches are replaced using a LRU replacement policy.

OC corresponds to an increase of [11%,31%] of the cache-hit ratio metric and a decrease of [9%,20%] of the hop-count metric against the alternatives for scenario I. OC has been shown to perform closely to ProbCache while LCE has been shown to perform the worst among the alternatives. The benefits of OC have been verified using the metric of cache evictions. To this end, OC has been shown to reduce the number of cache evictions at each level of the binary-tree topology, by one to two magnitudes of order. Thus, OC has been shown to successfully cache popular contents.

To examine the effect of parameter β on the performance of the caching policies, a range of β values have been considered, $\beta\epsilon[0.3, 1]$. The results suggest that the benefit of OC with regard to the rest of the alternatives, ranges between [1.5%,30%] for the cache-hit ratio metric and from less than 1% to 20% for the hop-count metric. In particular, OC performs the best among the alternatives if $\beta = 0.7$, while its performance degradates if $\beta > 0.7$. This is due to the fact that higher values of the parameter β will cause the probability of unpopular contents to be increased. In addition to the value of parameter β, a range of cache sizes, i.e., [5, 120] contents, have also been considered. The results suggest that an increase of the cache size corresponds to an improvement on the performance of all caching policies, while OC remains the optimal caching policy. In more detail, OC outperforms the rest of the alternatives by an increase of $[7.5\%, 28\%]$ of the cache-hit ratio metric and a decrease of $[1\%, 18\%]$ of the hop-count metric. The metric of cache evictions with regard to either the effect of parameter β or the effect of the cache size have not been examined.

The performance gains of OC with regard to scenario II, verify the dominance of this caching policy. For the purspose of this scenario, the range of cache sizes is set to be equal to $[40, 120]$ contents. Based on the results, OC yields to an increase of $[3\%, 35\%]$ of the cache-hit ratio metric and a decrease of $[9\%, 30\%]$ of the hop-count metric, respectively.

f) CRCache: CRCache [107] aims to optimize the utilization of caches by selectively caching contents to routers that are determined to be important. In order to achieve this, CRCache, equation (16), is composed of two factors: i.) the *Popularity* factor, equation (17) and ii.) the *RouterImportance* factor, equation (18). CRCache exploits the correlation between the two factors as follows: the higher this correlation, the higher the probability to cache content.

$$CRCache = 1 - AbsDiff\left(\frac{Map(Popularity)}{N}, \frac{Map(RouterImportance)}{M}\right) \quad (16)$$

$$Popularity = \begin{cases} CAP, & (Initialization) \\ Popularity + RC - \lambda \times \Delta T, & (Update) \end{cases} \quad (17)$$

$$RouterImportance = \alpha \times RID + (1 - \alpha) \times F \quad (18)$$

Due to the fact that the *Popularity* and *RouterImportance* factors are not directly comparable, CRCache is based on the unification of both factors. Both quantities are divided into a number of levels, N and M, respectively. According to Wang *et al.* [107], the determination of N is based on the content popularity distribution. The determination of M is based on the traffic patterns of a network and should be, only, set by an ISP. However, a description of the aforementioned decisions is missing. According to the definition of N and M, the value of each factor is mapped to the corresponding level and divided by the total number of levels. Then, the correlation betweeen the two factors is defined as the absolute difference of the factors while the caching probability is defined as the complementary probability of their correlation, equation (16).

The *Popularity* factor is calculated at each node on a delivery path and is equal to the frequency of content requests. Despite its own value, the *Popularity* factor may be updated according to the values of the variables of *Content Basic Popularity (CBP)* and *Content Attached Popularity (CAP)*. CBP is the content popularity being set by an ISP at the content source. The initialization of CBP may be based on either previous traffic patterns collected by the ISP or the metadata of the content. CAP is the content popularity being attached at both the content request and the content reply packet formats. CAP is set to be equal to the value of either the CBP or the Popularity factor, if the content request arrives to a content source or an intermediate node, respectively.

The *Popularity* factor is set to be equal to the value of CAP at the *Initialization Phase* and equal to the value of equation: $Popularity + RC - \lambda \times \Delta T$ at the *Update Phase*, equation (17). According to this equation, the *Popularity* factor is increased by a parameter RC and decreased by a parameter $\lambda \times \Delta T$. RC is additively increased upon the arrival of a content request and $\lambda \times \Delta T$ is used to guarantee that if the request frequency for a content has been reduced, the content will be subject to eviction. ΔT represents the time interval where content popularity calculations are being applied. More popular contents should correspond to a higher value of λ. All popularity calculations lie on the category of standalone popularity calculations (Section VII-D1).

The *RouterImportance* factor is calculated at each node on a delivery path using a weighted average equation, equation (18). In this equation, *RID* denotes the degree-centrality of a node and F denotes the traffic load of a node; the degree-centrality is a centrality-based metric that indicates the number of edges on a node and is described in Section VII-B3. The factor α denotes the weight coefficient of the equation.

At this point, it is important to highlight a number of observations. Firstly, the operation of CRCache implies that the value of variable *CAP* should be updated on either an intermediary node or a content source. Therefore, CRCache shares the same concerns with regard to the calculation of the variables x and y, as defined for ProbCache (Section VII-A2a). However, the computational cost in this case is expected to be lower, since CAP is updated only once, instead of every node on a delivery path. Secondly, the formulation of the *Popularity* factor lacks the consideration of multiple *CBP* values. Assuming a scenario where different ISPs may suggest different *CBP* values, a combination of these values is necessary upon the arrival of either content requests or content replies on a node. Last, a number of parameters lacks of the relevant description such as the determination of the variables N, M, while both variables depend on the collection of centralized information. This comes in contrast to the distributed nature of ICN networking.

To have an indication of the operation of CRCache, an example based on equation (16) is being provided. For the purpose of this example, the values of both variables N and M are set to equal to the values being used in the simulations, i.e., N=M=20;. For simplicity, the *Popularity* factor of content c is represented by the symbol P_c. The *RouterImportance* factor of node n is represented by the symbol RI_n. Assuming that the values of the variables $Map(P_c)$ and $Map(RI_n)$ at nodes: N_1, N_2, N_3, N_4 are equal to: $Map(P_c) = (5, 1, 3, 4)$ and $Map(RI_n) = (16, 4, 2, 20)$, respectively, the caching probability at each node will be equal to: $p = (0.45, 0.85, 0.95, 0.2)$.

Evaluation [107]: CRCache has been evaluated against BC, LCD, LCE and RND(p), using an arbitrary topology of about 8×10^4 nodes and a catalog size of 27×10^4 objects; BC is a centrality-based metric that indicates the frequency of which a node is included in the sets of shortest paths and is described in Section VII-B3. The caching operation is held on an object level. However, no information has been provided with regard to the distribution of the objects at the content sources or the distribution of the content sources and the content consumers within the network. The evaluation metrics include the cache-hit ratio and the hop-reduction ratio. A range of cache sizes, [0.01,100]Gb have been considered to highlight the importance of the cache size on the performance of the caching policies. The definition of a content replacement policy is missing.

CRCache has been shown to outperform the alternatives for both evaluation metrics. Even though, its benefits are almost negligible in cases where the cache size is lower than 1Gb, this difference increases as the cache size increases. In more detail, when the cache size is equal to 100Gb, the performance of CRCache against the rest of the alternatives is estimated to be an increase of [33%,66%] of the cache-hit ratio metric and an increase of [31%,56%] of the hop-reduction ratio metric.

The aforementioned benefits have been shown to be considerably affected by the factor of content popularity. To this end, a categorization of the values of content popularity has been conducted, where content popularity values are divided into N=20 equal groups, named *levels*. CRCache has been shown to be outperformed by both the LCD and BC, when the popularity level of contents is less than 14. This outcome verifies the dependency of CRCache on the factor of content popularity and its calculation. In more detail, this outcome verifies that CRCache favors, only, contents that are considerably popular.

B. Graph-Based Caching

Due to the integration of the caching and forwarding mechanisms in ICN networking, graph-based metrics may be used to decide on which node to cache content. Graph-based caching policies aim to react to the topology of a network by taking into account the nodes on a delivery path.

1) Edge Caching: Edge caching [79] has been proposed to push content closer to consumers by caching content on the last node on a delivery path. Edge caching aims to reduce the number of hops traversed towards a content source, therefore, reduce both the content delivery times and the traffic being introduced in a network due to the transmission of content requests. Even though, the goal of edge caching engages with the goal of in-network caching, a number of disadvantages may be withdrawn due to its functionality, that are described as follows. Firstly, content requests that are received on any node rather than the last node on a delivery path will be propagated towards a source. Depending on the network topology and a number of related assumptions such as

whether an AS or ISP is eligible to forward content requests to its peers or whether the stretch of the delivery paths is high or low, this effect may be either high or low, respectively. Secondly, since caching is applied on a single node on a delivery path, the cache capacity is considerably reduced. Due to this reason, edge caching may result in a degradation of the expected performance. The aforementioned claim has been further verified using a qualitative analysis [85]. Thirdly, edge caching fails to address the problem of cache pollution (Section VII-D).

Evaluation I [79]: Edge caching has been evaluated against LCE on both simulation and testbed experiments. The simulations have been based on a number of binary-tree topologies of $d\epsilon[2, 4]$ (Section IV-B4). A single consumer is placed at each node of the binary-tree. The evaluations have been conducted using cache sizes of [0%,100%] of the catalog size. However, neither the catalog size nor the popularity and content distributions have been defined. The naming granularity of the content unit is also being missing. Content is evicted from caches using a *First In-First Out (FIFO)* replacement policy. The evaluation metrics include the hop-count, responses per request and absorption times. The results conclude to a decrease of maximum 1 hop and 1 response per request in favor of LCE, compared to edge caching. The difference of the caching policies with regard to the metric of absorption times has been shown to be negligible.

The testbed experiments have been based on a 2-level binary-tree topology. Similar to the former evaluation scenario, a single consumer is placed at each node of the binary-tree. The content distribution and the naming granularity of the content unit remain unknown. The catalog size is set to be 40 contents. The evaluation metrics include the download times and the responses per request. LCE has been shown to outperform edge caching by an decrease of [10%,20%] of the download times metric and an decrease of 30% of the responses per request metric against edge caching.

Evaluation II [94]: More recently, edge caching has been evaluated against LCE using a range of simulation experiments. The simulations have been based on network topologies composed of the core network and the access network. The former has been constructed using Rocketfuel traces [50]. The latter has been constructed based on the assumption that each node represents the root of a 5-level binary-tree topology. Each root hosts a subset of the catalog size, defined in objects. Nevertheless, the definition of the catalog size is being missing. Consumers are placed at the leaf nodes of the binary-tree topologies. The cache size is set to be equal to 5% of the catalog size. Content is evicted from caches using a LRU replacement policy. The caching operation is held on an object level. The evaluation metrics include the server-hit ratio, hop-count ratio and maximum link traffic.

LCE may yield to a maximum benefit of 9% in the case that trace-driven parameters are used, i.e., if the Zipf parameter $\alpha = 1.04$ and up to 17% in the case where the simulation parameters are favorable to LCE, i.e., $\alpha = 0.1$. The evaluation results refer to all evaluation metrics. A higher benefit of the edge caching policy may be achieved if a double cache size is configured at all leaf nodes or if the peer nodes are eligible to cooperate with one another. The aforementioned conditions may reduce the difference in favor of LCE from 9% to 6%.

Evaluation III [85]: Edge caching has been evaluated against LCE using an arbitrary topology of 200 nodes and the metric of hop-count. The catalog size is set to be equal to 10^3 objects. The caching operation is held on an object level. All objects are uniformly distributed at random nodes within the network. Consumers issue object requests using a Zipf distribution of $\alpha = 1$, following either an IRM model or a temporal reference locality model (Section V-H). Cache sizes are set to be equal to $[10, 10^3]$ objects. Content is evicted from caches using a LRU replacement policy.

The results conclude to an increase of the hop-count metric for LCE compared to edge caching, as either the cache size or the localization factor l increases; the localization factor l defines both the *spacial locality* and the *time locality* of content requests. Both terms have been explicitly described in Section V-H. Nevertheless, this difference shows to decrease once the cache size reaches a threshold value. This threshold value is estimated to be equal to 50 objects when $l = 0$ and 10 objects when $l = 0.9$. The difference between the caching policies is estimated to be between [0, 1] hops.

The aforementioned results come in contrast to the ones that have been concluded by former evaluations for edge caching against LCE, i.e., evaluation I and evaluation II, that yield in favor of LCE. The reason for this contradiction is the variety of evaluation scenarios being used and the lack of representative network topologies; the majority of the experiments have been conducted using a binary-tree topology. As discussed in Section V-F, this assumption is invalid in ICN networking as both the content sources and consumers may be placed at various locations within a network. Secondly, the difference between the caching policies with regard to the metric of hop-count is at maximum 1. This indicates that edge caching and LCE are closely related to one another. In more detail, LCE always performs edge caching. Hence, the superiority of one caching policy against the other is more likely to be modest.

2) Leave-Copy Down (LCD): The *Leave-Copy Down (LCD)* [56] was initially proposed as an alternative to LCE (Section VII-A1c) for the Web caching problem [99]. LCD progressively caches the content on a delivery path, towards to the consumers. In order to achieve this, LCD pushes the content one hop closer to the consumers each time that a content request arrives. This way, LCD tries to capture the content popularity. Nevertheless the event of caching unpopular content is still likely to occur. This type of content popularity is named *path-based popularity* and is explicitly described in Section VII-D. An example of the operation of LCD is illustrated in Fig. 12. According to Fig. 12, all nodes will possess a content, similar to LCE, upon the arrival of four content requests. Even though, LCD may not yield to equal resource consumption to LCE, its functionality is rather similar. Thus, both caching policies share the same disadvantages. Last, LCD is based on the assumption that the node that serves the content request indicates to its neighbor node along the delivery path whether to cache a content using either a flag

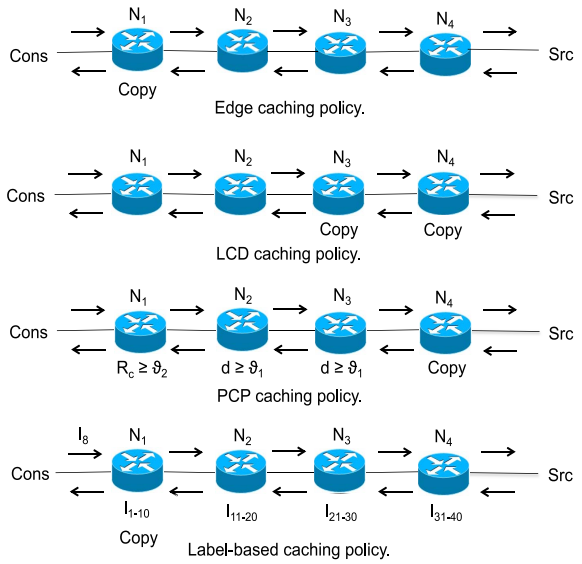


Fig. 12. Representation of non-probabilistic caching policies; graph-based and label-based caching policies.

or a hop-count metric, similar to y (Section VII-A2a); a flag indicates the caching status of a content. Consequently, similar to ProbCache (Section VII-A2a), LCD shares the same concerns with regard to the update of the content reply packets. The computational cost in this case is expected to be lower, since only the content reply packets are being updated.

Evaluation [56]: LCD has been evaluated against LCE, RND(0.75) and RND(0.9) under a number of Internet-like topologies, with a maximum number of 68 nodes [50]. A 3-level binary-tree topology has also been used as a benchmark. The evaluation has been based on a catalog size of 10^8 objects and a cache size of 10^6 contents or chunks. Thus, a chunk naming granularity is being defined. Object sizes follow a geometric distribution of a mean value of 10Mb. The chunk size is equal to 10Kb. Consumers issue object requests using a Zipf distribution of $\alpha = 1.5$. The evaluation metrics include the cache-hit ratio, hop-count ratio, load fairness and cache diversity. However, only the results of the cache-hit ratio are being presented in the paper. In order to explore the effect of a forwarding mechanism on the performance of the caching policies, two forwarding mechanisms have been considered: i.) a *single-path forwarding* mechanism and ii.) a *multi-path forwarding* mechanism. A single-path forwarding mechanism exploits the shortest routing paths or *primary paths*, calculated using the Dijkstra algorithm [108]. A multi-path forwarding mechanism exploits both the primary paths and the shortest routing paths that are most diversed from the primary paths.

The results suggest that regardless of the type of the forwarding mechanism, both the LCD and LCE conclude to an equivalent performance gain with regard to the metric of cache-hit ratio. This performance gain is slightly lower than the one concluded for RND(0.75) and RND(0.9). In addition to this, the performance gain with regard to either caching policy is higher when a single-path forwarding mechanism is used, compared to the multi-path forwarding mechanism. The aforementioned behavior may be explained as follows. Due to the parallel expression of content requests, a number of content replies will be sent over the equivalent delivery paths. These content replies will possibly cause a number of cache evictions on the nodes along the delivery paths, hence, reduce the utilization of caches. This phenomenon is referred to as *cache pollution* and is explicitly described in Section VII-D.

a) WAVE: WAVE [58] integrates the operation of LCD (Section VII-B2) with an exponentially increasing caching mechanism. Therefore, WAVE holds the same performance drawbacks derived from its ancestor. In contrast to the majority of caching policies that may be applied on either level of naming granularity (Section II-B), WAVE is strictly applied on a chunk naming granularity. The operation of WAVE may be defined as follows: upon the arrival of a request for an object, an increase on the number of chunks to be cached and the number of nodes that perform the caching occurs.

The operation of WAVE is illustrated in Fig. 13. According to Fig. 13, upon the first arrival of an object request, only the 1^{st} chunk of the object is cached one hop closer to the consumer, node N_4. Upon the second arrival of an object request for the same object, the next two chunks of the object are cached on node N_4 while the 1^{st} chunk is cached one hop closer, on node N_3. Upon the n^{th} arrival of an object request for the same object, 2^{n-1} chunks will be cached on node *n-1*, 2^{n-2} chunks will be cached on node *n-2*, etc; where n is the number of nodes on a delivery path. In order to achieve this functionality, the node serving the object request indicates to its neighbor node along the delivery path, which chunks to cache using a flag. A flag indicates the caching status of a chunk and is injected in the content reply's packet format. Chunks are cached if the flag equals to one and not otherwise.

In addition to the drawbacks derived from its ancestor, WAVE is subject to the following weakness. Since, the operation of WAVE is strictly related to the acknowledgement of object requests, WAVE limits the occurance of requests to object requests alone. In more detail, WAVE is unable to enhance the performance of a network, if the applications are interested in retrieving parts of objects rather than full objects.

Evaluation [58]: WAVE has been evaluated against LCE, RND(0.1) and UniCache under an Internet-like topology of 55 nodes, generated using the GT-ITM simulation tool [52]. The traffic model has been based on a catalog size of 10^5 objects, distributed uniformly over 10 randomly selected nodes. The caching operation is held on a chunk level. The object size is set to be equal to 1Gb and the chunk size is set to be equal to 10Mb. The cache size is set to be equal to 10Gb. Content is evicted from caches using a LRU replacement policy. Consumers issue object requests following a Zipf distribution of $\alpha = 0.85$. The simulation metrics include the cache-hit ratio, cache evictions, caching efficiency, hop-count and network traffic. According to Cho *et al.* [58] and in contrast to the description of cache evictions in Section IV-A2c, the metric of cache evictions in WAVE is equal to the number of chunks that have not experienced a cache-hit.

WAVE has been shown to correspond to an increase of 4% of the metric of cache-hit ratio against RND(0.1) and

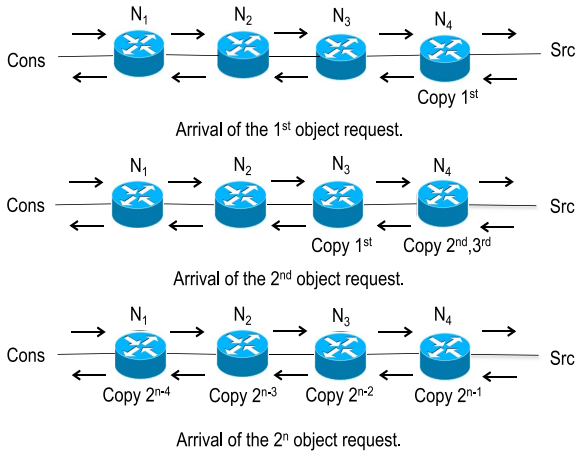


Fig. 13. WAVE caching policy.

UniCache and an increase of 12% of the same metric against LCE. WAVE has been also shown to exhibit the lowest eviction rates, with a difference of [44%,53%] compared to the alternatives. The caching efficiency of WAVE has been estimated to an increase of [94%,99%] of the caching efficiency values of the remaining caching policies. WAVE has been also shown to outperform the alternatives with regard to the hop-count and the network traffic metrics, by 1 hop and [10%,27%], respectively. The results verify the superiority of WAVE compared to the LCE, RND(0.1) and UniCache.

b) Progressive caching policy (PCP): The *Progressive Caching Policy (PCP)* [62] constitutes a combination of existing caching policies that are being applied depending on the position of a node within a network. According to this criterion, a node is categorized as an *immediate downstream node*; if it is the node after the one that has satisfied the content request towards a consumer(s), an *edge node*; if it is a node connected to consumers or an *intermediate node*; if it is neither an immediate nor an edge node. Upon the arrival of a content reply, an immediate node will cache a copy according to LCD (Section VII-B2). An intermediate node will cache a copy if the number of faces where the content requests for this content arrived, d, is larger or equal to the value θ_1, $d \geq \theta_1$. An edge node will cache a copy if the number of content requests for this content, R_c, is larger or equal to the value θ_2, $R_c \geq \theta_2$; this type of caching is named *static-popularity caching* and is explicitly described in Section VII-D2. This way, PCP tries to avoid caching unpopular contents. Despite the importance of both the θ_1 and θ_2 variables, no further details have been given with regard to their calculation. Last, PCP shares the disadvantages of both the LCD and the static-popularity caching due to its dependency on their functionality. The operation of PCP is displayed in Fig. 12.

Evaluation [62]: PCP has been evaluated against LCE using an arbitrary topology of 21 nodes and 2 servers that are arbitrarily placed within the network. The evaluation has been conducted under two traffic models of synthetic workloads, generated using the Prowgen simulation tool [109]. A chunk naming granularity is being defined. In the former, the object requests are issued using a Zipf distribution of $\alpha = 0.96$. In the latter, the object requests are issued using a Zipf distribution of $\alpha = 0.74$. The cache size for both workloads lies between $[1, 80]$Mb. The content size or chunk size is set to be equal to 10Kb. The catalog size and the object size have not been defined. The evaluation metric is the cache-hit ratio.

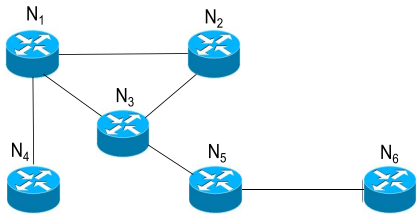


Fig. 14. Graph example used in the calculations of centrality-based caching policies.

PCP outperforms LCE by a maximum increase of 20% of the metric of cache-hit ratio for the intermediate nodes and a maximum increase of 50% of the same metric for the edge nodes, for either workload. This difference has been shown to be reduced to an equality as either the cache size of the intermediate nodes decreases or the cache size of the edge nodes increases. For the same reason, PCP is most likely to be outperformed by edge caching if sufficient cache capacity exists at the edge nodes. Last, the impact of parameter α on the performance of the caching policies is rather indistinguishable for the intermediate nodes while an increase of about 50% has been concluded for the edge nodes and the former workload.

3) Centrality-Based Caching: Rossini and Rossi *et al.* [110] have examined the suitability of a number of graph-related centrality-based metrics, including the *Betweenness Centrality (BC)*, *Closeness Centrality (CC)*, *Degree Centrality (DC)*, *Eccentricity Centrality (EC)*, *Graph Centrality (GC)* and *Stress Centrality (SC)* to determine the size of caches, e.g., proportional to the centrality value of the nodes, rather than determining the nodes to cache content. A Topology Manager (TM) is used to calculate and assign the centrality values of the nodes within a network. In this section, a description of the calculation of these metrics is being given. The calculations are based on the shortest-path routing mechanism, according to which the distance between a pair of nodes is measured in hop counts. However, alternative routing mechanisms may be used as well. The shortest-path distance between the nodes m and n is defined as: $d(m, n)$. For ease of understanding, a network is presented as a graph $G{:}(V, E)$, of $|V|$ vertices and $|E|$ edges.

Table III summarizes the sequence of nodes that are lying on the shortest paths between a pair of nodes, as presented in Fig. 14. In this table, symbol "–" indicates that the nodes are directly connected. As an example, the shortest-path between the nodes N_1 and N_6 is: $d(N_1, N_6) = (N_1, N_3, N_5, N_6)$. Based on the same topology, Table IV summarizes all centrality values.

a) Betweenness centrality (BC): The *Betweenness Centrality (BC)* metric indicates the frequency of which a node is included in the sets of shortest paths between all pairs of nodes and besides the current node n: $BC_n = \sum_{k,m \neq n}^{V} d(k, m, n)/d(k, m)$, where $d(k, m, n)$ is the

TABLE III
SHORTEST PATH CALCULATIONS BETWEEN NODES FOR THE GRAPH IN FIG. 14

	$\mathbf{N_1}$	$\mathbf{N_2}$	$\mathbf{N_3}$	$\mathbf{N_4}$	$\mathbf{N_5}$	$\mathbf{N_6}$
N_1	-	-	-	-	N_3	N_3, N_5
N_2	-	-	-	N_1	N_3	N_3, N_5
N_3	-	-	-	N_1	-	N_5
N_4	-	N_1	N_1	-	N_1, N_3	N_1, N_3, N_5
N_5	N_3, N_1	N_3	-	N_3, N_1	-	-
N_6	N_5, N_3	N_5, N_3	N_5	N_5, N_3, N_1	-	-

TABLE IV
REPRESENTATION OF THE CENTRALITY VALUES OF NODES FOR THE GRAPH IN FIG. 14

	BC	**CC**	**DC**	**EC**	**GC**	**SC**
N_1	4/10	1/3	3	2	1/2	4
N_2	0	1/4	2	2	1/2	0
N_3	6/10	1/2	3	1	1	6
N_4	0	1/7	1	3	1/3	0
N_5	4/10	1/5	2	2	1/2	4
N_6	0	1/8	1	3	1/3	0

shortest-path distance between the nodes k and m that passes through n. The number of shortest paths that include node N_1 equals to four. The number of shortest paths between all pairs of nodes, besides N_1 equals to ten. Thus, $BC_{N_1} = 4/10$.

b) Closeness centrality (CC): The *Closeness Centrality (CC)* metric refers to the reverse sum of the shortest-path distances between a node n and the remaining nodes in a graph, defined as: $CC_n = 1/\sum_{m \neq n}^{V} d(m, n)$. The shortest-path distances between node N_1 and the remaining nodes, i.e., N_2, N_3, N_4, N_5, N_6 are: $(0, 0, 0, 1, 2)$, respectively. Accordingly, $CC_{N_1} = 1/(0 + 0 + 0 + 1 + 2) = 1/3$.

c) Degree centrality (DC): The *Degree Centrality (DC)* metric indicates the number of connections or edges of a node n: $DC_n = deg(n)$. As an example, $DC_{N_1} = 3$.

d) Eccentricity centrality (EC): The *Eccentricity Centrality (EC)* metric refers to the maximum shortest-path distance between a node n and the remaining nodes in a graph: $max_{\forall m \epsilon V \neq n} d(m, n)$. The maximum shortest-path distance between node N_1 and the remaining nodes equals to two. Consequently, $EC_{N_1} = 2$.

e) Graph centrality (GC): The *Graph Centrality (GC)* metric refers to the maximum, reverse, shortest-path distance between a node n and the remaining nodes in a graph. Consequently, GC equals to the reverse EC value: $1/max_{\forall m \epsilon V \neq n} d(m, n)$. As an example, $GC_{N_1} = 1/2$.

f) Stress centrality (SC): The *Stress Centrality (SC)* metric indicates the times that a node is included in the sets of shortest paths between all pairs of nodes, besides the current node n: $SC_n = \sum_{k,m \neq n}^{V} d(k, m, n)$. The number of shortest paths that include node N_1 equals to four. Hence, $SC_{N_1} = 4$.

Chai *et al.* [73] proposed a caching policy based on the BC metric that may be defined as follows: upon the delivery of a content reply, a copy is cached on the node(s) with the highest BC value. To this end, a field that holds the highest BC value of the nodes being traversed on the forwarding path is injected in the content request's packet format. This value is then injected in the equivalent field of the content reply's packet format. Thus, the aforementioned caching policy shares the same concerns with regard to the update of both the content request and the content reply packet formats, as described in ProbCache (Section VII-A2a). Even though, the caching policy has been defined using the BC metric, any other centrality-based metric may be used instead. As an example, assuming that the delivery path of a content is equal to the shortest-path between the nodes N_1 and N_6, $d(N_1, N_6) = (N_1, N_3, N_5, N_6)$ and according to the information presented in Table IV, a replica will be cached on node N_1, if the caching decision is based on either the BC, CC or GC metric, on nodes N_1 and N_3, if the caching decision is based on the DC metric and on node N_6, if the caching decision is based on the EC metric.

A disadvantage that derives from the nature of the centrality-based metrics is their limited scalability due to their centralized operation; centralized information is required for their operation, collected by, e.g., a TM. Consequently, centrality-based caching policies are improper to be applied on large-scale networks such as the Internet. Even under an optimistic scenario where ISPs are engaged to exchange information with regard to their connectivity, the scalability of a global TM would be, still, questionable. DC constitutes an exception to this rule. Moreover, the performance of centrality-based caching policies may be considerably affected by the network topology and its connectivity. The aforementioned claim may be further explained through an example. For the purpose of this example, caching is applied on the nodes with the highest DC value on a delivery path; similar to the operation of the caching policy that is based on the BC metric and has been described earlier. For the rest of this section, we will refer to this caching policy as the *DC-based Caching Policy (DCCP)*. If the connectivity of a network is uniform, i.e., if the DC metric of the majority of nodes within a network is equivalent, the performance of DCCP will be equivalent to the performance of LCE. In more detail, the same copy is expected to be cached by the

majority of nodes within a network, resulting in high resource consumption and high cache redundancy (Section VII-A1c). In addition to the above and based on the fact that the nodes that are closer to the core of a network experience higher connectivity than the nodes that are closer to the consumers, centrality-based caching policies may be characterized by the tendency to cache content on the former category of nodes rather than the edge nodes. This comes in contrast to a number of caching policies that aim to push content closer to consumers to reduce the content delivery times and the network traffic, such as the ProbCache (Section VII-A2a), ProbPD (Section VII-A2d) and edge caching (Section VII-B1). This tendency is expected to be higher for DCCP. Last, centrality-based caching policies do not take into account the factor of content popularity.

Evaluation I [110]: The aforementioned centrality-based metrics have been evaluated under a number of Internet-like topologies of 11 to 68 nodes [50], using the evaluation metrics of cache-hit ratio and hop-count ratio. The evaluation has been based on a catalog size of 10^8 objects. Object sizes follow a geometric distribution of a mean value of 10Mb. Cache sizes are set to be 10Gb. Consumers issue object requests using a Zipf distribution of $\alpha = \{1.25, 1.5\}$. The caching operation is held on a chunk level. However, the content size or chunk size and the cache eviction policy have not been defined. In addition to this, no information has been given with regard to the distribution of the objects to the content sources within the network or the content consumers within the network.

According to the conclusions, DC is the most effective centrality-based metric compared to the alternatives with regard to both evaluation metrics. However, no numerical evidence have been provided to support this conclusion while the corresponding figures are rather hard to interpret.

Evaluation II [73]: BC has been evaluated against LCE and UniCache under a 5-level binary-tree topology, a scale-free topology of 100 nodes [69] and an Internet-like topology of 6804 nodes [51]. The evaluation has been conducted using a catalog size of 10^3 contents. The cache size is set to be equal to 10^2 contents. However, the naming granularity of the content unit has not been defined. Content is evicted from caches using a LRU replacement policy. Consumers issue content requests using a Zipf distribution of $\alpha = 1$. The evaluation metrics include the server-hit ratio and hop-count ratio. Similar, to the previous evaluation, no information has been given with regard to the distribution of the contents to the content sources within the network or the content consumers within the network.

The results suggest a reduction of 33% and 29% of the server-hit ratio metric in favor of BC for the binary-tree topology, against LCE and UniCache, respectively. BC has been also shown to outperform its alternatives with regard to the metric of hop-count ratio for the same network topology; a reduction of the range of $[2\%, 15\%]$ has been concluded. Since the results of BC against LCE and UniCache are approximately identical for both the scale-free and the Internet-like topology, only the former results are being described. In more detail, the benefit of BC against the alternatives is estimated to be between $[9\%, 26\%]$ and 8% with regard to the metrics of server-hit ratio and hop-count ratio, respectively.

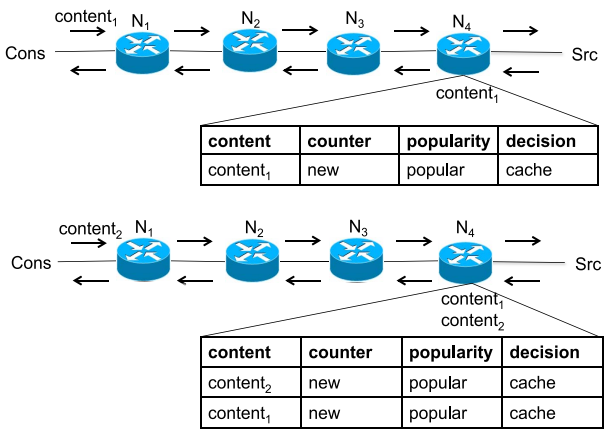
C. Label-Based Caching

Label-based caching [53] has been propose to increase the content diversity of a network and provide sparse content distribution. In label-based caching, the nodes are eligible to cache a specific range of contents. To make this decision, each node is assigned a range of labels, defined a priori. Each node is also aware of the range of labels of the remaining nodes within a network. Thus, label-based caching constitutes a form of cooperative caching (Section V-E). The operation of label-based caching may be explained as follows. If a content's label falls in the range of labels for which a node on a delivery path is responsible, the content is cached. If otherwise, the content is forwarded to the node being responsible. To perform this decision, each node on a delivery path is obligated to execute a hash function. This function may be applied on either the content identifier or the content's sequence number, where a sequence number represents the chunk number of an object.

The modulo hash function is the simplest form of hash functions and is based on the use of sequence numbers. The modulo hash function is formulated as follows: *seq mod k*, where *seq* is the content's sequence number and *k* is a divisor set a priori. Similar to WAVE, label-based caching may only be applied on a chunk naming granularity. The operation of label-based caching is illustrated in Fig. 12. Assuming that the label of a content request is equal to 8 and the label ranges on nodes: N_1, N_2, N_3, N_4 are equal to: [1, 10], [11, 20], [21, 30], [31, 40], respectively, a copy of the content will be cached on node N_1.

Despite its advantages, label-based caching may also yield to a number of disadvantages. In more detail, since nodes are responsible for a specific range of contents, both the caching decisions and the contents are bound to a specific topological position, set a priori. Thus, label-based caching limits the generation of copies to one copy alone, in the scale where the caching policy is being applied, while increases the likelihood for sub-optimal forwarding decisions [27], [53]. In addition to this, label-based caching is based on the existence of a centralized configuration module that maintains full knowledge of the topology and the traffic patterns of a network [27], [111]. Consequently, label-based caching is limited to the scale of an ISP or an AS. Due to its centralized operation and its pre-defined behavior, label-based caching contradicts the distributed nature and the dynamic environment of ICN networking. Finally, the utilization of hash functions may significantly increase the complexity and the computational cost on intermediate nodes, while sequence-based hash functions can not be applied when no sequence numbers are used.

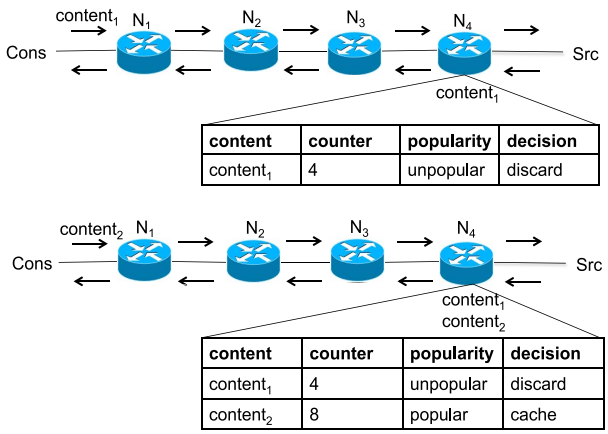
Evaluation [53]: Label-based caching has been evaluated against LCE using an Internet-like topology of 87 nodes [50] and a catalog size of 54×10^3 contents or chunks. Thus, a chunk naming granularity is being defined. The chunk size is equal to 7.5Mb. A number of 5 content sources, that possess all contents, are arbitrarily placed within the network. A number of 200 consumers are uniformly placed at the edge nodes of the network. Each node is equipped with a cache size of 1Gb.



content	counter	popularity	decision
$content_1$	new	popular	cache

content	counter	popularity	decision
$content_2$	new	popular	cache
$content_1$	new	popular	cache

Fig. 15. Representation of path-based popularity caching policies: example of the LCD caching policy (Section VII-B2).



content	counter	popularity	decision
$content_1$	4	unpopular	discard

content	counter	popularity	decision
$content_1$	4	unpopular	discard
$content_2$	8	popular	cache

Fig. 16. Representation of static-popularity caching policies.

Content is evicted from caches using a LRU replacement policy. The evaluation has been based on the metrics of download times, server hits and cache diversity. The content popularity distribution has not been defined.

Considering a range of k values, $k\epsilon[1, 6]$, label-based caching corresponds to a reduction of $[30\%, 75\%]$ of the metric of server hits against LCE. Label-based caching has been also shown to outperform its alternative by a maximum increase of 60% of the cache diversity metric. In contrast to its advantages, label-based caching suggests an increase of 15% of the metric of download times of LCE. This difference is equal to 40ms. The aforementioned results verify our assumptions with regard to the benefits of label-based caching.

D. Popularity-Based Caching

Measurement studies on in-network caching technologies, such as Web caching and on-path caching [56], [83], [112], have highlighted the dependency of caching policies on the frequency and distribution of content requests. The *cache pollution* problem, caused due to *one-timer* objects [112], has been shown to affect the performance of a caching policy by increasing both the download times and the network traffic [40], [113]. One-timer objects are objects that have been requested once, while cache pollution is the case where one-timer objects are cached. One-timer objects have been estimated to lie between 45% and 75% of the content requests [112]. Since on-path caching is expected to serve a higher number of objects than replication technologies, under more severe constraints such as the cache size (Section V-B), cache pollution becomes an important complication that a variety of caching policies such as the fixed-probabilistic caching policies (Section VII-A1) and the graph-based caching policies (Section VII-B) fail to address.

To overcome this problem, *popularity-based caching* policies have based their decision on content popularity [58], [95], [105]. The idea behind popularity-based caching is that popular contents will satisfy a higher number of content requests. Thus, caching popular contents should be preferred while caching unpopular contents should be prevented. Depending on the scale that is used to determine content popularity, path or node, content popularity may be categorized into: i.) *path-based popularity* and ii.) *local-based popularity*, respectively. In the former, caching is applied on at minimum one node on a delivery path upon the arrival of the first content request. In the latter, no such guarantee exists; caching is applied based on the content popularity observed on a node. Path-based popularity fails to address the cache pollution problem effectively.

LCD (Section VII-B2) and *WAVE* (Section VII-B2a) are applied on a path scale and implicitly compare the popularity of a content against the remaining contents. To ease understanding, Fig. 15 illustrates the operation of LCD, where $content_1$ and $content_2$ are cached on node N_4, upon the arrival of a corresponding content request. The rest of the nodes on the delivery path do not perform a caching decision. Since no local information is used, the value of the *popularity counter* for each content is set to *"new"* and the popularity status for a "new" content is set to *"popular"*; the term popularity counter is explicitly described in local-based popularity calculations (Section VII-D1). The term has been broadly used to ensure consistency among the figures. Besides LCD and WAVE, no additional caching policies have been proposed with regard to path-based popularity calculations. Therefore, for the rest of this paper, we concentrate mainly on the description of local-based popularity calculations.

Depending on the method that is used to define content popularity: a standalone popularity counter or a comparison of this counter against either a static or a dynamic value, local-based popularity may be further categorized into: i.) *standalone local-based popularity*, ii.) *static local-based popularity* and iii.) *dynamic local-based popularity*, respectively. For the rest of this paper, we will refer to the caching policies that integrate such popularity decisions as: i.) *standalone-popularity caching*, ii.) *static-popularity caching* and iii.) *dynamic-popularity caching*, correspondingly. A description of the aforementioned popularity calculations and caching policies may be found in the following subsections.

1) Standalone-Popularity Caching: In standalone-popularity caching, content popularity is determined using a *popularity counter*. In its simplest form, a popularity counter is additively increased upon the arrival of a content request

and decreased through time. This is in order to ensure that if the number of requests for a content has been reduced, its popularity counter will be reduced accordingly; the content will be subject to eviction. Depending on a number of criteria that may be considered to be important for the calculation of content popularity, a wide range of equations, named *update equations*, may be constructed to update a popularity counter. As an example, CRCache (Section VII-A2f) decreases its popularity counter by a rate: $\lambda \times \Delta T$, where λ is a constant, set by an ISP and ΔT is the time interval for which the content popularity calculations are being performed. An additional example of standalone-popularity caching is decsribed in the following subsection, Section VII-D1a.

According to the existing literature review, the validity of standalone popularity calculations has not been, yet, explored. In more detail, the selection of the parameters that may update a popularity counter has been rather left to be arbitrarily decided. To this end, CacheShield (Section (VII-D1a)) and OC (Section (VII-A2e)) have only considered the definition of an additively update equation. This verifies the claim that the definition of an update equation is challenging.

a) CacheShield: CacheShield [114] has been proposed as a caching policy able to improve the performance of a cache server or an ICN node and prevent cache pollution attacks. For the purpose of this paper, only the details of the deployment of CacheShield on an ICN node are being discussed.

CacheShield aims to prevent the occurrence of locality-disruption attacks [115], according to which consumers request unpopular contents to weaken the locality of caches and degradare their performance. CacheShield is based on the assumption that legitimate content requests follow a Zipf distribution while malicious content requests follow a uniform distribution. Hence, locality-disruption attacks may be prevented by preventing unpopular contents from being cached.

To this end, CacheShield has been based on the use of *content names*. A content name is defined as the combination of a content identifier and a popularity counter R_c. Content names are stored in the cache of a node and are subject to all relevant operations, such as cache eviction. Upon the arrival of a content reply, a node is able to make a caching decision using a logistic function, equation (19), formulated by the popularity counter R_c and the parameters p and q, that define the logistic function. If the logistic function implies that the content should not be cached, only its popularity counter is being updated. If otherwise, the content is cached while its content name is being replaced. According to the authors, further research on the selection of the parameters p and q is essential.

$$Popularity = \frac{1}{1 + e^{\frac{(p-R_c)}{q}}} \tag{19}$$

To have an indication of the operation of CacheShield, Fig. 19 illustrates an example. In this example, the parameters p and q are set to be equal to: $p=20$ and $q=1$, respectively; the same values have been used for the evaluation of CacheShield. To ease calculations, R_c is set to be equal to: $R_c = (5, 3, 8, 4)$, on nodes: N_1, N_2, N_3, N_4, respectively. Consequently, the caching probability with regard to the same nodes is equal to: $p = (1/1 + e^{15}, 1/1 + e^{17}, 1/1 + e^{12}, 1/1 + e^{16})$.

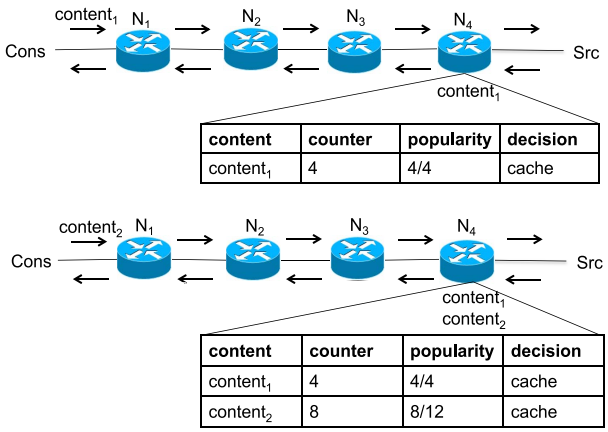
Evaluation [114]: CacheShield has been evaluated against LCE under a number of evaluation scenarios: i.) *scenario I*, ii.) *scenario II* and iii.) *scenario III*. All scenarios are subject to the following assumptions: Each content source shares an equal amount of contents, i.e., 10^5 contents; however, the naming granularity of the content unit has not been defined. All nodes are equipped with a cache capacity of 1% of the catalog size. Content is evicted from caches using a LRU replacement policy. Consumers issue content requests following a Zipf distribution of $\alpha = 0.9$. The evaluation is conducted using the metric of cache-hit ratio. The values of the parameters p and q are set to be equal to: p=20 and q=1, respectively. At this point, it is important to highlight that for the purpose of this paper, only the evaluation of CacheShield against LCE, under legitimate traffic, is being considered.

Scenario I consists of a 4-node string topology and a single content source that is placed at one end of the network topology. A total number of thee consumers are arbitrarily placed within the network. The catalog size in this scenario is equal to 10^5 contents. Scenario II constitutes an extension of scenario I, where an additional content source is placed at the other end of the network topology. The catalog size in scenario II is equal to 2×10^5 contents. Scenario III consists of an arbitrary topology of nine nodes. A total number of three content sources and six consumers are arbitrarily placed within the network. The catalog size equals to 3×10^5 contents.

CacheShield outperforms its alternative by an increase of $[28\%, 36\%]$ and $[33\%, 67\%]$ of the cache-hit ratio metric, for scenario I and scenario II, respectively. The benefit of CacheShield over LCE for scenario II, with regard to the same evaluation metric has been estimated to be equal to 13%. The results verify the importance of the popularity factor into the caching decision and on the performance of a caching policy.

2) Static-Popularity Caching: In static-popularity caching, popularity calculations require the definition of a threshold *thr*. Contents with a number of requests higher than *thr* are considered to be popular while contents with a number of requests lower than *thr* are considered to be unpopular [40], [62], [89]. Unpopular contents are excluded from the caching decision. Due to the volatile nature of ICN networking, we expect the definition of a threshold to be challenging, resulting in out of date calculations and unutilized cache capacity [40], [113]. An example of static popularity calculations is defined by PCP (Section VII-B2b). Fig. 16 illustrates the operation of static-popularity caching where the popularity counter of $content_1$ is equal to 4, while the popularity counter of $content_2$ is equal to 8. Assuming that *thr=5*, $content_1$ is considered to be unpopular, hence, it is discarded, while $content_2$ is considered to be popular, hence, it is cached. In contrast to the path-based popularity calculations, local-based popularity calculations are performed on each node on a delivery path. Consequently, a similar operation will be performed on nodes N_1, N_2 and N_3. An additional example of static-popularity caching is decsribed in the following subsection, Section VII-D2a.

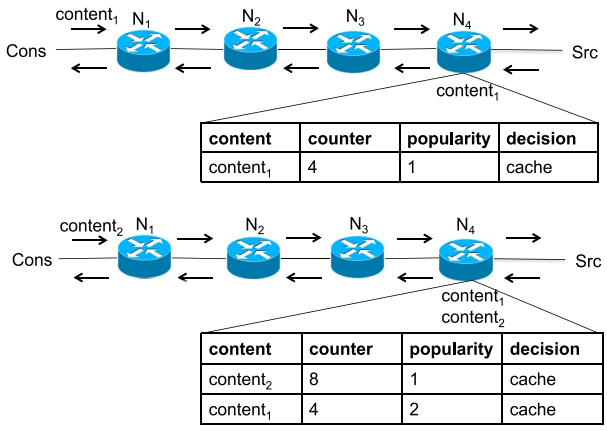
a) Most popular content (MPC): Similar to PCP (Section VII-B2b), *Most Popular Content (MPC)* [102] is based on static popularity calculations. MPC differs from PCP



content	counter	popularity	decision
content$_1$	4	4/4	cache

content	counter	popularity	decision
content$_1$	4	4/4	cache
content$_2$	8	8/12	cache

Fig. 17. Representation of implicit dynamic-popularity caching policies.



content	counter	popularity	decision
content$_1$	4	1	cache

content	counter	popularity	decision
content$_2$	8	1	cache
content$_1$	4	2	cache

Fig. 18. Representation of explicit dynamic-popularity caching policies.

in the following aspects. Firstly, MPC aims to distribute popular content towards all the neighbors of a node rather than the neighbors on the delivery path(s), alone. In order to achieve this, MPC distributes notification messages, named *suggestion messages*. A suggestion message is an extended version of a content reply packet. However, no details of the exact format of this packet have been given. Secondly, upon the reception of a suggestion message, a neighbor is eligible to decide whether to cache the suggested content according to its local caching policy. As an example, a node may discard the suggested content if not enough cache capacity exists. Lastly, to prevent the repeated transmission of suggestion messages for the same content, MPC has introduced the notion of a *reset value (rv)*. A reset value is used to reset the value of the popularity counter, after the transmission of the suggestion message has occured.

To have an indication of the operation of MPC, Fig. 19 illustrates an example, where $thr = 5$. The values of the popularity counters, R_c are set to be equal to the ones being used for CacheShield (Section VII-D1a). Suggestion messages are illustrated via a red arrow. According to MPC, a copy is cached on the nodes N_1 and N_3, while a suggestion message is sent to their neighbors, i.e., nodes N_2 and N_4. At this point, it is worth noting that node N_2 will receive a duplicate suggestion message. Such a behavior may significantly increase the traffic of a network, if the connectivity of the network is high.

Evaluation [102]: MPC has been evaluated against LCE using the metrics of cache-hit ratio, hop-reduction ratio, caching frequency and cache diversity. To examine whether the performance of MPC is affected by a number of parameters, such as the catalog size and the cache size, two evaluation scenarios: i.) *scenario I* and ii.) *scenario II* have been considered. Scenario I consists of a 3-level binary-tree topology and a catalog size of 10^4 objects. The caching operation is held on a chunk level. The object size is set to be equal to 1Mb. The cache size is set to be equal to 10^3 contents or chunks. Content is evicted from caches using a LRU replacement policy. Scenario II consists of a range of Internet-like topologies of 11 to 68 nodes [50] and a catalog size of 10^8 objects. The object size is set to be equal to 10Mb. The cache size is set to be equal to 10^6 contents or chunks. Once again, content is evicted from caches using a LRU replacement policy. Regardless of the network topology being used, a single content server and a total number of eight consumers are randomly placed within the network. Consumers issue object requests using a Zipf distribution of $\alpha = 1.5$. The chunk size is set to be equal to 10Kb. Both scenarios have been based on the same values of the variables thr and rv, i.e., thr=5 and rv=0, respectively. The aforementioned values have been concluded after an extensive number of simulations, held under the binary-tree topology and the set-up of scenario II. This conclusion verifies our concerns with regard to the efficiency of static popularity calculations.

The results suggest a decrease of $[0, 38\%]$ of the caching frequency metric of MPC against LCE. The size of the decrease increases as the value of parameter α increases, $\alpha = \{0.5, 1.0, 1.5, 2.0, 2.5\}$. Suprisingly, no additional results have been provided with regard to the metrics of hop-reduction ratio and cache diversity, while the evaluation results with regard to the metric of cache-hit ratio are lacking of the corresponding interpretation. Hence, the results of the cache-hit ratio metric, for scenario I, are intentionally being omitted.

According to the results for scenario II, MPC corresponds to an increase of $[3.5\%, 12\%]$ of the cache-hit ratio metric and a decrease of $[10\%, 75\%]$ of the caching frequency metric, against LCE. The variation of the caching frequency depends on the nodes' connectivity; the more connected they are, the higher this metric will be. In contrast to the aforementioned results, MPC has been shown to be outperformed by its alternative using the metric of cache diversity. Depending on the network topology, a decrease of 70% to approximately 90% is shown. This conclusion verifies the assumption that MPC caches less contents than LCE due to the fact that MPC caches only popular contents. Similar to scenario I, the results for the metric of hop-reduction ratio have not been provided.

3) Dynamic-Popularity Caching: Dynamic-popularity caching policies refer to the popularity of a content with regard to the popularity of the remaining contents within a network, during a time interval ΔT [41], [89], [95]. A common technique is to define an infinite time interval [58], [95]. Dynamic popularity may be applied either implicitly or explicitly, i.e., i.) *implicit dynamic-popularity* and ii.) *explicit dynamic-popularity*, respectively. In the

former, the content popularity is defined as the fraction of content requests for a content to the total number of content requests [95], [105], thus, being a probability $\epsilon[0, 1]$. In the latter, this comparison is achieved by constructing a decreasingly ordered list of contents based on their number of requests [89], [113], [116]. At this point, it is worth noting that the number of requests for a content is represented by the popularity counter for this content, R_c. Then, a content's popularity is determined depending on its position in this list. Explicit dynamic-popularity calculations require the definition of a threshold *thr*. However, the use of a threshold *thr* in this case is different to static popularity. A threshold value of *thr=10* suggests that the first ten contents of this ordered list are popular while the remaining contents are unpopular.

As an example, *ProbPD* (Section VII-A2d) is based on implicit dynamic-popularity calculations while PBLRU (Section VII-D3a) is based on explicit dynamic-popularity calculations. Fig 17 and Fig. 18 illustrate the operation of implicit dynamic-popularity calculations and explicit dynamic-popularity calculations, respectively. In the former, the content popularity is determined by dividing the number of requests for a content to the total number of content requests. Since $content_1$ is the only content that has been requested, its popularity may only be compared against itself. Hence, $content_1$ is concluded to be popular and being cached. When a caching decision for $content_2$ is made, its content popularity is determined against the total number of content requests, hence, being equal to 8/12. Since this probability is still reasonably high, $content_2$ is eventually being cached. In the latter, the content popularity is determined by the position of contents in a decreasingly ordered list, that is constructed according to the number of requests for all contents. For the purpose of this example, *thr=2*, meaning that only the first two contents of this ordered list are popular. Once again, since $content_1$ is the only content that has been requested, it lies on the top of this list. Therefore, $content_1$ is considered to be popular and is being cached. When a decision for $content_2$ is made, the list of contents'popularity is reconstructed. Since, $content_2$ corresponds to a higher number of content requests, it is listed first, while $content_1$ is now listed second. However, since *thr=2*, both contents are defined to be popular, despite of the number of content requests that each of them has received. Consequently, $content_2$ is also being cached.

Due to the formation of implicit dynamic-popularity calculations, a considerable number of content requests is necessary to differentiate a content from the remaining contents and be defined as popular. This complication results in a degradation of the overall content popularity. An important factor to this effect is the definition of the time interval ΔT; small time intervals are less likely to conclude to an overall view of the contents' popularity. This assumption has been verified through a thorough evaluation on implicit dynamic-popularity calculations [117]. According to the results, the factor of catalog size may equally affect the efficiency of implicit dynamic-popularity calculations as follows: smaller catalog sizes result in a clearer pattern of content requests, thus, more representative implicit dynamic-popularity calculations.

Explicit dynamic-popularity calculations avoid the occurence of the aforementioned drawback by constructing a decreasingly ordered list based on the popularity counters of contents. To this end, a sorting algorithm is executed each time that the popularity counter of a content is being updated. The execution of a sorting algorithm results in an increase of both the computational resources and the content delivery times. Due to the dependency on the definition of a threshold *thr*, explicit dynamic-popularity calculations share the same disadvantages to static popularity calculations (Section VII-D2).

Dynamic popularity may be further categorized depending on the observations being used for its calculation, i.e., the current observation alone, extracted during the time interval ΔT or a combination of observations extracted during various time intervals. According to this criterion, the categories of: i.) *single-based popularity* and ii.) *multi-based popularity* are being defined. According to the literature review of this paper, all the caching policies that consider content popularity as a caching criterion, use single-based popularity calculations. To this end, Janaszka *et al.* [113] proposed a caching policy based on multi-based popularity calculations. A description of this policy is given in the following subsection, Section VII-D3a.

a) Popularity-based least-recently used (PBLRU): The *Popularity-Based Least-Recently Used (PBLRU)* [113] caching policy bases its caching decision on the popularity of content determined using the formula of Exponential Moving Average (EMA) and the LRU replacement policy. This formula considers both the current popularity of a content R_c, concluded by updating its popularity counter and the former popularity of the content, $Popularity_{\Delta T_p}$, concluded using the equation (20).

$$Popularity_{\Delta T} = Popularity_{\Delta T_p} \times (1 - \alpha) + \alpha \times R_c \quad (20)$$

The $Popularity_{\Delta T}$ is then used to determine the categorization of contents into one of the three groups, i.e., i.) the *Popularity Group I (PG-I)*, ii.) the *Popularity Group II (PG-II)* and iii.) the *Popularity Group III (PG-III)*, where PG-I corrseponds to the most popular contents, PG-II corresponds to medium popular contents and PG-III corresponds to the unpopular contents. PBLRU caches a content if it belongs in group PG-I and not othewise. At this point, it is worth noting that even though the definition of two groups alone, would be sufficient to define this distinction, this content categorization is also being used to determine the forwarding of content requests, thus, it is linked to its functionality.

PBLRU belongs in the category of explicit dynamic-popularity calculations. As a result, the aforementioned categorization is based on a decreasingly ordered list consists of the number of requests for each content and the definition of thresholds. An example of the aforementioned functionality has been already given in Fig. 18. In addition to Fig. 18, Fig. 19 summarizes the calculations of equation (20), where $\alpha = 0.5$ and $Popularity_{\Delta T_p} = 1$. The values of the popularity counters, R_c are set to be equal to the ones being used for CacheShield (Section VII-D1a). The $Popularity_{\Delta T}$ on nodes: N_1, N_2, N_3, N_4, is: $P = (3, 2, 4.5, 2.5)$, respectively.

Evaluation [113]: PBLRU has been evaluated against LCE, under a 5-level diamond topology of 7 nodes (Section IV-B2)

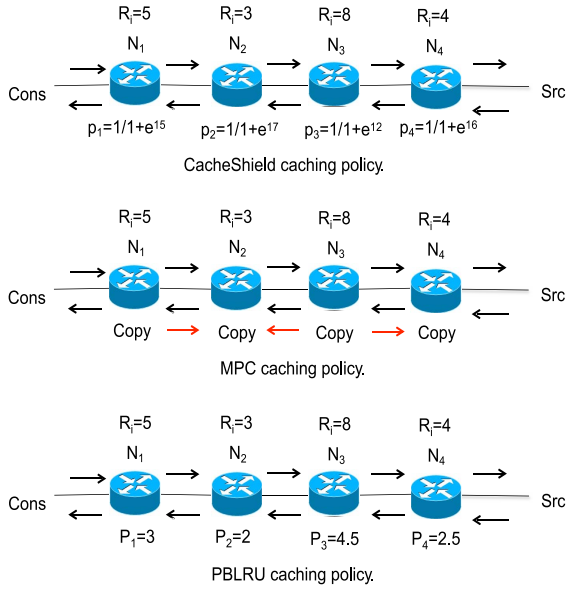


Fig. 19. Representation of popularity-based caching policies.

and an arbitrary topology of 13 nodes, using the metrics of cache-hit ratio and download times. In the former, a consumer is placed at one end of the network topology and a content source is placed at the other. In the latter, each node is connected to both, a consumer and a content source. A content source holds a subset of objects, which may be either equal to the remaining sources or determined based on the objects' popularity distribution. The evaluation has been based on a catalog size of 10^3 objects and a cache size of 30Mb. The caching operation is held on a chunk level. The object size is set to be equal to 1Mb. The content size or chunk size is set to be equal to 10Kb. Content is evicted from caches using a LRU replacement policy. Object requests are issued following a Zipf distribution of $\alpha = \{0.2, 0.4, 0.6, 0.8, 1\}$.

PBLRU outperforms LCE by an increase of [5%, 26%] of the cache-hit ratio metric and a maximum decrease of 0.1 seconds of the download times metric for the diamond topology, while this difference increases to 3 seconds for the arbitrary topology. The results of the cache-hit ratio metric for the arbitrary topology have not been presented in the paper. Once again, the results verify the impact of the popularity factor on the performance of a caching policy.

VIII. Discussion

The purpose of this section is to provide a discussion on the observations being made with regard to the existing caching policies. The dicsussion aims to highlight the current trends and evaluation issues of caching policies in ICN networking, maintain a qualitative comparison with regard their advantages and disadvantages and suggest directions for future work.

A. Evolution of Caching Policies

According to the information provided in Section IV-C1, existing caching policies may be categorized depending on the caching criteria used. Fig. 20 illustrates the evolution of the caching criteria used in the literature, where the right column refers to the caching policies and the left column refers to the caching criteria used by a caching policy; a bullet in the right column corresponds to its symmetric bullet in the left column.

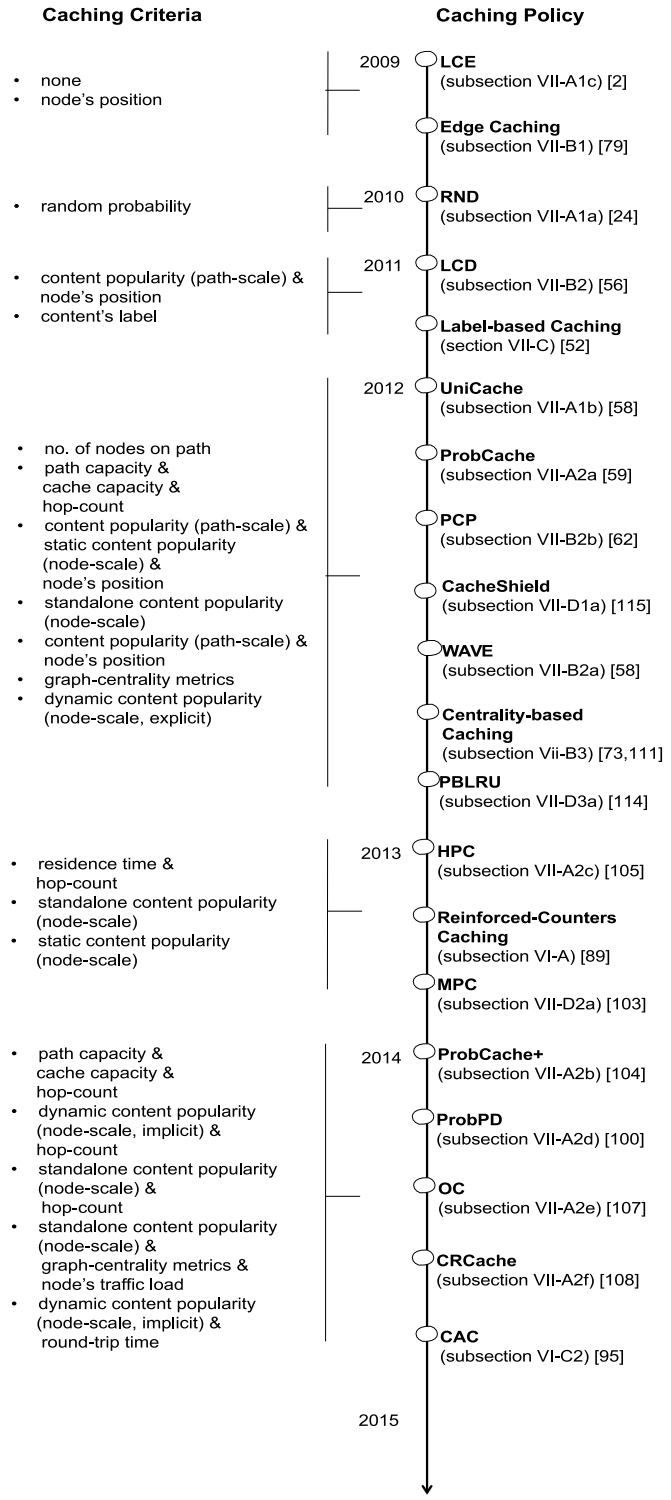


Fig. 20. Time-based categorization and criteria analysis of the existing caching policies.

Consulting Fig. 20, caching policies have been evolved from simplistic implementations that consider either topological information, such as the position of a node on a delivery

TABLE V
TAXONOMY OF THE PROPOSED CACHING POLICIES AND THE CACHING POLICIES BEING CONSIDERED IN THEIR EVALUATION, THE CACHING MODEL, CACHING LEVEL OF OPERATION AND TRAFFIC MODEL CHARACTERISTICS

Proposed Policy	Comparison Policy	Caching Model	Caching Level	Catalog Size	Cache Size	Content Popularity
BC [73]	LCE, UniCache	centralized	-	10^3 content units	10^2 content units	$\alpha = 1$
CAC [95]	BC, LCE	cooperative	chunk	10^4 content units	500 content units	$\alpha = 0.8$
CacheShield [114]	LCE	autonomous	-	$\{1, 2, 3\} \times 10^5$ content units	$\{1, 2, 3\} \times 10^3$ content units	$\alpha = 0.9$
CRCache [107]	BC, LCE, LCD, RND(p)	autonomous	object	27×10^4 objects	$[0.01, 100]Gb$	-
BC, CC, DC, EC, GC, SC [110]	BC, CC, DC, EC, GC, SC	centralized	chunk	10^8 objects	10Gb	$\alpha = \{1.25, 1.5\}$
Edge caching [79]	LCE	autonomous	-	40 content units	$[4, 40]$ content units	-
	LCE		object	-	5% catalog size	$\alpha = 1.04$
	LCE		object	$\{10^2, 10^3, 10^7\}$ objects	$[10, 10^3]$ objects	$\alpha = 1$
HPC [104]	LCE, ProbCache	cooperative	chunk	$\{85, 850\}$ objects	200Kb	$\alpha = 0.85$
Label-based [53]	LCE	cooperative	chunk	54×10^3 chunks	1Gb	-
LCD [56]	LCE, RND(0.75), RND(0.9)	autonomous	chunk	10^8 objects	10Gb	$\alpha = 1.5$
LCE [2]	-	autonomous	packet	-	-	-
MPC [102]	LCE	autonomous	chunk	$\{10^4, 10^8\}$ objects	$\{0.01, 10\}$Gb	$\alpha = \{0.5, 1, 1.5, 2, 2.5\}$
OC [106]	LCE, MPC, ProbCache, RND(0.3), RND(0.7)	cooperative	packet	$\{1, 7\} \times 10^3$ packets	$[5, 120]$ packets	$\alpha = 0.73$
PBLRU [113]	LCE	autonomous	chunk	10^3 objects	30Mb	$\alpha = \{0.2, 0.4, 0.6, 0.8, 1\}$
PCP [62]	LCE	autonomous	chunk	-	$[1, 80]Mb$	$\alpha = \{0.74, 0.78, 0.92, 0.96\}$
ProbCache [59]	LCE, LCD, RND(0.3), RND(0.7)	cooperative	-	-	[1,6] seconds	$\alpha = 0.8$
ProbCache+ [103]	Edge-caching, LCE, LCD, ProbCache	cooperative	-	$[10^3, 10^4]$ cache size	[1,6] seconds	$\alpha = 1.2$
ProbPD [105]	DC, LCE, ProbCache, RND(0.9)	cooperative	-	10^3 content units	10^2 content units	$\alpha = 1$ & $\alpha = 1.5$
RND(p) [24]	LCE	autonomous	packet	1 objects	-	-
WAVE [58]	LCE, RND(0.1), UniCache	cooperative	chunk	10^5 objects	10Gb	$\alpha = 0.85$
Reinforced-Counters caching [89]	-	autonomous	-	-	-	-

path or arbitrary information, such as the use of random probabilities, into advanced implementations that consider network characteristics and content characteristics such as the RTT and the content popularity, respectively. According to the same figure, the criteria of hop-count and content popularity have been broadly used in the literature while a considerable number of caching policies have based their functionality on both criteria, combined using a dynamic probability. Examples of such caching policies are the: CRCache (Section VII-A2f), ProbPD (Section VII-A2d) and OC (Section VII-A2e).

At this point, it is worth noting that former caching policies that have considered content popularity as a caching criterion have based their decision on either path-based, static or standalone-popularity calculations. Latter caching policies have based their decision on dynamic-popularity calculations. According to the description of popularity-based caching policies (Section VII-D), each category of popularity calculations has its own advantages and disadvantages while no quantitative comparison exists against one another. Hence, a conclusion on which approach is preferable can not be made.

TABLE VI
TAXONOMY OF THE PROPOSED CACHING POLICIES AND THE CACHING POLICIES BEING CONSIDERED IN THEIR EVALUATION, THE TRAFFIC MODEL CHARACTERISTICS, EVALUATION METRICS AND NETWORK TOPOLOGIES. THIS TABLE CONSITITUES A CONTINUATION OF TABLE V

Proposed Policy	Comparison Policy	Object Size	Chunk Size	Evaluation Metrics	Network Topology
BC [73]	LCE, UniCache	-	-	hop-count ratio, server-hit ratio	5-level binary-tree topology & 100-nodes scale-free topology [69] & 6804-nodes CAIDA topology [51]
CAC [95]	BC, LCE	$[0.5, 3.0]Mb$	-	hop-count ratio, download times, cache-hit ratio	10×10 grid topology, 100-nodes scale-free topology [69], various Rocketfuell topologies [50], 100-nodes MANET topology of 10 clusters
CacheShield [114]	LCE	-	-	cache-hit ratio	4-nodes string topology & 9-nodes arbitrary topology
CRCache [107]	BC, LCE, LCD, RND(p)	-	-	hop-count ratio, cache-hit ratio	8×10^4-nodes arbitrary topology
BC, CC, DC, EC, GC, SC [110]	BC, CC, DC, EC, GC, SC	$G(10Mb)$	-	hop-count ratio, cache-hit ratio	11 to 68 nodes Rocketfuel topologies [50]
Edge caching [79]	LCE	-	-	hop-count, download times, responses per request, absorption times	2-level to 4-level binary-tree topologies
	LCE	-	-	hop-count ratio, maximum link traffic, server-hit ratio	Rocketfuel topology [50] for the core network & 5-level binary-tree topology for the access networks
	LCE	-	-	hop-count	200-nodes arbitrary topology
HPC [104]	LCE, ProbCache	$[1, 100]Kb$	1Kb	cache-hit ratio, cache redundancy, absorption times	7-nodes string topology & 8-nodes arbitrary topology
Label-based [53]	LCE	-	7.5Mb	download times, server hits, cache diversity	87-nodes Rocketfuel topology [50]
LCD [56]	LCE, RND(0.75), RND(0.9)	$G(10Mb)$	10Kb	hop-count ratio, load fairness, cache-hit ratio, cache diversity	11 to 68 nodes Rocketfuel topologies [50]
LCE [2]	-	-	-	-	-
MPC [102]	LCE	$\{1, 10\}$Mb	10Kb	hop-count ratio, cache-hit ratio, caching frequency, cache diversity	3-level binary-tree topology & 11 to 68 nodes Rocketfuel topologies [50]
OC [106]	LCE, MPC, ProbCache, RND(0.3), RND(0.7)	-	-	hop-count, cache-hit ratio, cache evictions	3-level binary-tree topology & 52-nodes Rocketfuel topology [50]
PBLRU [113]	LCE	1Mb	10Kb	download times, cache-hit ratio	7-nodes diamond topology & 13-nodes arbitrary topology
PCP [62]	LCE	-	10Kb	cache-hit ratio	21-nodes arbitrary topology
ProbCache [59]	LCE, LCD, RND(0.3), RND(0.7)	-	-	hop-count ratio, server hits, cache evictions	6-level binary-tree topology
ProbCache+ [103]	Edge-caching, LCE, LCD, ProbCache	-	-	hop-count ratio, server hits, cache hits, cache evictions	200-nodes scale-free topology [69]
ProbPD [105]	DC, LCE, ProbCache, RND(0.9)	-	-	download times, cache hits, cache evictions	5-level binary-tree topology
RND(p) [24]	LCE	-	-	cache-hit ratio, download times	8-nodes string topology
WAVE [58]	LCE, RND(0.1), UniCache	1Gb	10Mb	hop-count ratio, network traffic, cache-hit ratio, cache evictions, caching efficiency	55-nodes GT-ITM topology [52]
Reinforced-Counters caching [89]	-	-	-	-	-

B. Evaluation of Caching Policies

According to the information presented in Table V and Table VI, a number of observations may be extracted. The first of the observations relates to the evaluation topologies being used (Table VI). According to Table VI, a considerable number of caching policies have been evaluated using either

a string topology or a binary-tree topology. Examples of such caching policies are the HPC (Section VII-A2c) and RND(p) (Section VII-A1a). As discussed in Section V-F, such types of topologies are not considered to be representative examples of network topologies. Even though, their use may provide some indication about the performance of a caching policy, the conclusions derived should not be generalized.

A complementary observation with regard to the use of evaluation topologies is the use of arbitrary topologies. According to Table VI, the scale of arbitrary topologies lies broadly between 8 and 20 nodes. Examples of caching policies that have used such evaluation topologies are the CacheShield (Section VII-D1a) and PBLRU (Section VII-D3a). Since, this scale is considerably low to be compared with the scale of the Internet, such types of topologies are not considered to be representative examples of networks topologies. Consequently, the conclusions derived from the corresponding evaluations should not be generalized to large-scale network topologies.

The second observation derived from Table V and Table VI relates to the substantially different values of the parameters being used in the evaluation of the existing caching policies. This mismatch complicates the comparison between the policies. Noticeable examples of this tendency are the catalog size and cache size parameters, both of which may considerably affect the performance of a caching policy [85], [94], [117].

Complementing the previous observation, a set of evaluation parameters have, also, being omitted. Noticeable examples of omitted parameters are the object size and chunk size parameters. Similar to the catalog size and cache size parameters, both the object size and the chunk size parameters may significantly affect the operation of a caching policy. As an example, if the object size equals to 10Mb and the chunk size equals to 10Kb, an object is interpreted to be equal to 10^3 chunks. If the object size equals to 10Gb, yet, the chunk size remains unchanged, an object is interpreted to be equal to 10^6 chunks. In the former, a caching decision is made over 10^3chunks. In the latter, a caching decision is made over the double number of chunks which reduces the caching probability by 50%. The aforementioned observation has been shown to significantly affect the operation of dynamic-popularity caching policies such as the ProbPD [117] (Section VII-A2d).

The last observation relates to the comparison caching policies, according to which newer caching policies have been evaluated. Based on the information extracted from either Table V or Table VI, LCE has been used as a benchmark in the evaluation of all newer caching policies while in some cases it is the only comparison policy being used. The evaluation of newer caching policies against a range of alternative caching policies could provide a better understanding on their performance and the effect of the parameters acknowledged in their evaluation. Therefore, the selection of a comparison caching policy should be based on the correlation of the proposed caching policy to the existing ones. As an example, PCP (Section VII-B2b) and WAVE (Section VII-B2a) are based on LCD. However, neither of them has been evaluated against it. Hence, no conclusion can be withdrawn if the newer caching policies suggest a benefit against their processor.

Based on the aforementioned observations and the fact that caching policies have been shown to be sensitive to a number of parameters, such as the cache size [85], [94], [118], catalog size [117] and network topology [73], [110] and dependable on a variety of features such as the forwarding mechanism [56], [88], [95] and the namespace of content identifiers [53], [111], a cross comparison between the existing caching policies, reviewed in this paper, is rather impractical to be made. To this end, a qualitative comparison that highlights the advantages and disadvantages of the existing caching policies is presented in the following subsection, Section VIII-C.

C. Qualitative Comparison of Caching Policies

In this subsection, a qualitative comparison of the existing caching policies, reviewed in this paper, is being given. For this purpose, Table VII and Table VIII provide a summary of the existing caching policies, consists of: the goal that a caching policy aims to fulfill, its advantages and disadvantages. Table VIII constitutes a continuation of Table VII. To ensure consistency between the tables, the first column of Table VII is duplicated to Table VIII. Similar to Tables V–VI, the symbol *"-"* is used to indicate that no conclusion has been drawn with regard to a category. As an example, the symbol *"-"* is used to indicate that ProbCache does not provide any advantages, since the caching policy fails to fulfill its aim. An important point that is worth noting is that in contrast to Tables V–VI, Tables VII–VIII consists of an additional caching policy, UniCache (Section VII-A1b). The reason to this difference is that UniCache is only listed as a comparison policy. Hence, its evaluation is included in the evaluation of the corresponding caching policies such as the WAVE (Section VII-B2a).

According to Table VII and Table VIII, fixed-probabilistic caching policies, such as the LCE (Section VII-A1c) and RND(p) (subsectionr̃efsec:random), are individually executed and involve no cooperation between the nodes within a network. Therefore, fixed-probabilistic caching policies are easy to demploy and introduce no additional overhead in a network. On the contrary, fixed-probabilistic caching policies neglect the variable nature of content requests and network topologies, resulting in high resource consumption and cache redundancy.

In order to address these problems, centrality-based caching policies (Section VII-B3) and label-based caching policies (Section VII-C), have based their decision on the selection of nodes within a network to perform caching. An important drawback of both caching policies is their reliance on global network knowledge. In addition to this, centrality-based caching policies tend to cache content closer to sources. Furthermore, an increase of both the cache redundancy and the resource consumption may be yield for uniformly connected networks. Centrality-based caching policies require the update of both the content request and the content reply packet formats, while label-based caching policies require the execution of a hash function upon the arrival of either a content request or a content reply. Consequently, both caching policies introduce an additional overhead and computational cost on nodes

TABLE VII
TAXONOMY OF THE PROPOSED CACHING POLICIES, THEIR PURPOSE, ADVANTAGES AND DISADVANTAGES

Proposed Policy	Purpose/Goal	Advantages/Issues Tackled	Disadvantages/Challenges
BC [73]	Exploit centralized nodes within a network to cache content	• Position distinction	• Global network knowledge • No content distinction • Tendency to cache closer to sources • Resource consumption/cache redundancy increase in uniformly connected networks • Update of content request/content reply packets (computational overhead)
CAC [95]	Avoid congested links by caching popular content at their edge	• Location distinction • Content distinction • Avoidance of congested links	• Inacurrate popularity estimations • Definition of time interval ΔT • Update of content request/content reply packets (computational overhead)
CacheShield [114]	Prevent cache pollution attacks by caching popular content	• Content distinction	• Definition of an update equation
CRCache [107]	Cache popular content T nodes that are determined to be important	• Location distinction • Content distinction	• Negligence of multiple popularities • Lack of parameters' calculation • Global network knowledge • Update of content request/content reply packets (computational overhead) • Definition of an update equation
BC,CC,DC,EC,GC,SC [110]	Exploit centralized nodes within a network to cache content	• Position distinction	• Global network knowledge • No content distinction • Tendency to cache closer to sources
Edge caching [79]	Cache content closer to consumers	• Reduction of hops traversed • Simplicity	• No content distinction • Tendency to cache on specific nodes
HPC [104]	Progressively cache content towards consumers for a specific time-interval	-	• Global network knowledge • No content distinction • Lower caching probability towards consumers (unfulfilled goal) • Inaccurate MRT_m estimations
Label-based [53]	Increase content diversity within a network	• Increase of content diversity	• Global network knowledge • A priori determined caching positions • A priori determined content to cache • Restiction to one copy per scale • Sub-optimal forwarding decisions • Execution of hash functions (computational overhead)
LCD [56]	Progressively cache content towards consumers	• Simplicity	• Cache redundancy • Resource consumption • No content distinction • Update of content reply packets (computational overhead)
LCE [2]	Cache content on all traversed nodes	• Simplicity • Low overhead • Fast content dissemination	• Cache redundancy • Resource consumption • No content distinction • No position distinction
MPC [102]	Cache popular content	• Content distinction	• Definition of a threshold

within a network, while they neglect the factor of content popularity. Finally, label-based caching policies increase the likelihood to sub-optimal forwarding decisions.

ProbCache (Section VII-A2a) and ProbCache+ (Section VII-A2b) try to tackle these problems from a different point of view. Their goal is to fairly share the

TABLE VIII
TAXONOMY OF THE PROPOSED CACHING POLICIES, THEIR PURPOSE, ADVANTAGES AND DISADVANTAGES. THIS TABLE CONSITUTES A CONTINUATION OF TABLE VII

Proposed Policy	Purpose/Goal	Advantages	Disadvantages/Challenges
OC [106]	Progressively cache popular content towards consumers	• Content distinction • Position distinction • Cache redundancy reduction	• Global network knowledge • Update of content request/content reply packets (computational overhead) • Definition of an update equation
PBLRU [113]	Cache popular content	• Content distinction	• Execution of a sorting algorithm • Definition of thresholds
PCP [62]	Progressively cache popular content towards consumers	• Content distinction	• Cache redundancy • Resource consumption • Update of content reply packets (computational overhead) • Definition of thresholds
ProbCache [59]	Fair allocation of capacity resources on a delivery path among contents	-	• No content distinction • Update of content request/content reply packets (computational overhead) • Unfair allocation of capacity resources (unfulfilled goal) • Arbitrary definition of parameters
ProbCache+ [103]	Fair allocation of capacity resources on a delivery path among contents	• Fair allocation of capacity resources	• No content distinction • Update of content request/content reply packets (computational overhead) • Arbitrary definition of parameters
ProbPD [105]	Progressively cache popular content towards consumers	• Content distinction • Position distinction	• Inacurrate popularity estimations • Definition of time interval ΔT • Update of content request/content reply packets (computational overhead)
Reinforced-Counters caching [89]	Cache popular content	• Content distinction	• Definition of an update equation
RND(p) [24]	Randomly cache content	• Simplicity • Low overhead	• No content distinction • No position distinction • Unpredictable random nature
UniCache [58]	Cache content at one node of a delivery path	• Simplicity • Low overhead	• No content distinction • No position distinction • Favors low-stretch delivery paths • Cache redundancy on short-stretch delivery paths
WAVE [58]	Progressively cache content towards consumers using an exponential mechanism	• Fast content disemination	• Cache redundancy • Resource consumption • No content distinction • Update of content reply packets (computational overhead) • Applicable to object requests alone

capacity resources of a delivery path among the contents. To this end, all contents will have a fair chance to be cached, depending on their distance from the consumers and the source. ProbCache+ has extended the functionality of ProbCache, due to the fact that ProbCache has been unable to fulfill its goal. However, both caching policies share the following drawbacks. Firstly, both caching policies are based on a number of parameters that have been arbitrarily set. Secondly, both caching policies necessitate the update of both the content request and the content reply packet formats,

thus, introducing additional computational cost on the nodes within a network. Thirdly, neither caching policy considers the factor of content popularity as a caching criterion, making no distinction on the cached content.

Towards a different direction, edge caching (Section VII-B1) aims to reduce the content delivery times and the traffic rates within a network by caching content on the last node of a delivery path. Edge caching is simple to deploy. Yet, it results in single points of caching. Due to this reason, the cache capacity of a delivery path is lower than the alternatives. Since the cache capacity of a node has been shown to be bound by an upper limit (Section V-B), the performance of edge caching is limited to the size of the caches [85]. Moreover, edge caching neglects the factor of content popularity.

Following the example of edge caching, a number of caching policies have been proposed towards caching the content closer to consumers, such as the HPC (Section VII-A2c). HPC aims to push content towards to consumers and cache it for a specific time interval. In order to achieve this, HPC is based on the mean residence time of contents in a cache, MRT_m. However, mean values do not constitute a representative metric in cases where the distribution function of the sample is neither uniform nor normal, while HPC fails to push the content closer to consumers. Besides the aforementioned drawbacks, HPC requires the provision of global network knowledge. Last, HPC does not take into account the factor of content popularity.

To this end, the rest of the existing caching policies, being reviewed in this paper, base their caching decision on the factor of content popularity; either explicitly or in a combination with additional caching criteria such as the hop-count metric. LCD (Section VII-B2) tries to capture the content popularity by caching content one hop closer to consumers upon the arrival of a content request. LCD constitutes an example of path-based caching policies (Section VII-D). Path-based caching policies are unable to address the cache pollution problem effectively (Section VII-D). WAVE (Section VII-B2a) and PCP (Section VII-B2b) are based on the functionality of LCD. Consequently, both caching policies share the shortcomings of cache redundancy and resource consumption. Nevertheless, WAVE and PCP are subject to a number of additional disadvantages that are described as follows. Due to its operation, WAVE stricts the naming granularity of requests to object requests alone while PCP requires the definition of thresholds; regardless of being a path-based caching policy, PCP is also a static-based caching policy in the sense that it relies on both path-based and static-based content popularity calculations. Last, both caching policies necessitate the update of content reply packets, thus, increase the computational overhead on the nodes within a network. In contrast to PCP, MPC (Section VII-D2a) relies on static-popularity calculations alone. Therefore, MPC concludes to a single disadvantage which is the definition of a threshold.

Similar to MPC, CacheShield (Section VII-D1a) and PBLRU (Section VII-D3a) are caching policies that are based on the criterion of content popularity alone. CacheShield is based on standalone-popularity calculcations (Section VII-D1). PBLRU is based on dynamic-popularity calculations (Section VII-D3). Both caching policies inherit the shortcomings derived from the popularity category to which they belong. CacheShield requires the definition of an update equation to update the popularity counters. PBLRU requires the definition of thresholds and the execution of a sorting algorithm to update the sorting lists of contents. Consequently, PBLRU introduces an additional computational overhead on the nodes within a network while CachShield does not.

CAC (Section VI-C2), ProbPD (Section VII-A2d), OC (Section VII-A2e) and CRCache (Section VII-A2f) are examples of caching policies based on the combination of the factor of content popularity with additional caching criteria. Regardless to CAC, that aims to avoid congested links by caching popular content at the edge of the congested links, the remaining caching policies aim to cache popular content either closer to consumers or at nodes that are determined to be important. CAC and ProbPD are based on implicit dynamic-popularity calculations (Section VII-D3). OC and CRCache are based on standalone-popularity calculations (Section VII-D1). Depending on the additional caching criteria used and the operation of the caching policies, a caching policy may yield to a number of additional weaknesses. OC and CRCache require global network knowledge for their operation and CRCache neglects the consideration of multiple content popularities and the definition of a number of parameters. Finally, all caching policies are based on the update of both the content request and content reply packet formats. Thus, introduce more overhead on the nodes within a network.

D. Directions to Future Work

According to the existing literature in ICN networking and on-path caching in particular, the determination of evaluation scenarios is impeded by an ongoing discussion around the validity of a number of parameters and the provision of sufficient tools to support long-scale evaluations.

As discussed in Sections IV-B5 and VIII-B, evaluation topologies such as string topologies, binary-tree topologies or small-scale arbitrary topologies are improper to be considered as representative network topologies for the Internet. On the contrary, large-scale topologies have been shown to require a considerable amound of time to be evaluated [118] using the existing simulations tools, such as the ndnSIM simulator [119]. Alternative simulation platforms have been argued to be less efficient than ndnSIM, such as the ccnSim [120] and the mini-CCNx [121]. Towards this direction, ongoing efforts on the evaluation of caching policies constitute the development of large-scale simulation tools [118] and the development of evaluation frameworks, able to provide a thorough analysis of the performance of caching policies [122]. Considering the emergence of ICN networking in today's Internet, further research on this diresction is essential in order to provide the community with sufficient simulation and analytical tools.

In addition to this, the determination of a number of evaluation parameters is impeded by an ongoing discussion around their validity. As discussed in Sections VII-D and V-H, the validity of a Zipf distribution for the characterization of the

objects' popularity [78] and the validity of the IRM model [85] for the characterization of the content requests have been questioned. Moreover, the use of a Poisson distribution for the arrival of content requests has been argued to be invalid [123]. Since the aforementioned parameters are vital to the evaluation of future network mechanisms, such as the on-path caching policies, able to affect their performance and operation [56], [83], [112], further research on their validity is essential.

Last, a cross comparison between the existing caching policies and newer caching policies should individually being promoted [124]. To this end, evaluation parameters such as the catalog size and the cache size should not be omitted while common evaluation parameters should be used when feasible. Our paper contibutes on this direction by focusing on a detailed description of the evaluation of the existing caching policies including a number of criteria such as the traffic model, evaluation metrics and network topologies used. Moreover, the evolution of the caching criteria and the effect of such criteria on the performance of a network are being discussed while one can observe the caching policies and parameters that have been used as benchmarks.

IX. Conclusion

In this paper, we have presented a survey of the caching problem in ICN, with a focus on on-path caching. In more detail, a description of the existing caching policies and the forwarding mechanisms that complement these policies has been provided. The description is enhanced via the use of examples. The caching policies have been categorized against their properties using a number of criteria such as the caching model and level of operation and the evaluation parameters used in their evaluation, to derive a taxonomy for on-path caching and highlight the trends and evaluation issues in this area. A discussion driven by the advantages and disadvantages of the existing caching policies and the challenges and open questions in on-path caching is finally being held.

References

[1] H. Balakrishnan *et al.*, "A layered naming architecture for the Internet," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Architect. Protocols Comput. Commun. (SIGCOMM)*, Portland, OR, USA, 2004, pp. 343–352.

[2] V. Jacobson *et al.*, "Networking named content," in *Proc. 5th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, Rome, Italy, Dec. 2009, pp. 1–12.

[3] J. Gantz, "The expanding digital universe: A forecast of worldwide information growth through 2010," White Paper, IDC, Mar. 2007.

[4] C. Index, "Cisco visual networking index: Forecast and methodology 2012-2017," White Paper, CISCO, May 2013.

[5] R. Buyya and M. Pathan, *Content Delivery Networks*, vol. 9, 1st ed. Heidelberg, Germany: Springer-Verlag, 2008.

[6] S. Androutsellis-Theotokis and D. Spinellis, "A survey of peer-to-peer content distribution technologies," *ACM Comput. Surveys (CSUR)*, vol. 36, no. 4, pp. 335–371, Dec. 2004.

[7] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst. (TOCS)*, vol. 2, no. 4, pp. 277–288, Nov. 1984.

[8] A. Passarella, "A survey on content-centric technologies for the current Internet: CDN and P2P solutions," *J. Comput. Commun.*, vol. 35, no. 1, pp. 1–32, Jan. 2011.

[9] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Commun. Surveys Tuts.*, vol. 7, no. 2, pp. 72–93, 2nd 2005.

[10] A. Vakali and G. Pallis, "Content delivery networks: Status and trends," *IEEE Internet Comput.*, vol. 7, no. 6, pp. 68–74, Nov./Dec. 2003.

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

[12] G. Xylomenos *et al.*, "A survey of information-centric networking research," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 1024–1049, 2nd Quart., 2014.

[13] C. Fang, F. R. Yu, T. Huang, J. Liu, and Y. Liu, "A survey of energy-efficient caching in information-centric networking," *IEEE Commun. Mag.*, vol. 52, no. 11, pp. 122–129, Nov. 2014.

[14] G. Zhang, Y. Li, and T. Lin, "Caching in information centric networking: A survey," *J. Comput. Netw.*, vol. 57, no. 16, pp. 3128–3141, Nov. 2013.

[15] G. Garcia *et al.*, "Comet: Content mediator architecture for content-aware networks," in *Proc. 20th Future Netw. Mobile Summit (FutureNetw)*, Warsaw, Poland, Jun. 2011, pp. 1–8.

[16] N. Fotiou, D. Trossen, and G. C. Polyzos, "Illustrating a publish-subscribe Internet architecture," *Telecommun. Syst.*, vol. 51, no. 4, pp. 233–245, Dec. 2012.

[17] C. Dannewitz *et al.*, "Network of information (NetInf)—An information-centric networking architecture," *Comput. Commun.*, vol. 36, no. 7, pp. 721–735, Apr. 2013.

[18] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surveys*, vol. 35, no. 2, pp. 114–131, Jun. 2003.

[19] J. Choi, J. Han, E. Cho, T. Kwon, and Y. Choi, "A survey on content-oriented networking for efficient content delivery," *IEEE Commun. Mag.*, vol. 49, no. 3, pp. 121–127, Mar. 2011.

[20] G. Tyson, N. Sastry, R. Cuevas, I. Rimac, and A. Mauthe, "A survey of mobility in information-centric networks: Challenges and research directions," in *Proc. 1st Workshop Emerg. Name Orient. Mobile Netw. Design Archit. Algorithm. Appl. (NoM)*, Hilton Head, SC, USA, Jun. 2012, pp. 1–6.

[21] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel, "Separating key management from file system security," in *Proc. 17th ACM Symp. Oper. Syst. Principles (SOSP)*, Charleston, SC, USA, Dec. 1999, pp. 124–139.

[22] M. Walfisha, H. Balakrishnana, and S. Shenker, "Untangling the Web from DNS," in *Proc. 1st Symp. Netw. Syst. Design Implement. (NSDI)*, San Francisco, CA, USA, Mar. 2004, Art. no. 17.

[23] A. Ghodsi *et al.*, "Information-centric networking: Seeing the forest for the trees," in *Proc. 10th ACM Workshop Hot Topics Netw. (HotNets)*, Cambridge, MA, USA, Nov. 2011, Art. no. 1.

[24] S. Arianfar, P. Nikander, and J. Ott, "On content-centric router design and implications," in *Proc. 2nd Workshop Re-Architect. Internet (ReARCH)*, Philadelphia, PA, USA, Nov. 2010, Art. no. 5.

[25] D. Perino and M. Varvello, "A reality check for content centric networking," in *Proc. 1st Workshop Inf. Centric Netw. (ICN)*, Toronto, ON, Canada, Aug. 2011, pp. 44–49.

[26] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and B. Mathieu, "A survey of naming and routing in information-centric networks," *IEEE Commun. Mag.*, vol. 50, no. 12, pp. 44–53, Dec. 2012.

[27] L. Saino, I. Psaras, and G. Pavlou, "Hash-routing schemes for information centric networking," in *Proc. 3rd ACM SIGCOMM Workshop Inf. Centric Netw. (ICN)*, Hong Kong, 2013, pp. 27–32.

[28] M. Dräxler and H. Karl, "Efficiency of on-path and off-path caching strategies in information centric networks," in *Proc. 2nd IEEE Int. Conf. Green Comput. Commun. (GreenCom)*, Besançon, France, Nov. 2012, pp. 581–587.

[29] Y. Thomas and G. Xylomenos, "Towards improving the efficiency of ICN packet-caches," in *Proc. 10th IEEE Int. Conf. Heterogen. Netw. Qual. Rel. Security Robustness (QShine)*, Rhodes, Greece, Aug. 2014, pp. 186–187.

[30] M.-J. Montpetit, C. Westphal, and D. Trossen, "Network coding meets information-centric networking: an architectural case for information dispersion through native network coding," in *Proc. 1st Workshop Emerg. Name Orient. Mobile Netw. Design Architect. Algorithm. Appl. (NoM)*, Hilton Head, SC, USA, Jun. 2012, pp. 31–36.

[31] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[32] J. K. Sundararajan *et al.*, "Network coding meets TCP: Theory and implementation," *Proc. IEEE*, vol. 99, no. 3, pp. 490–512, Mar. 2011.

[33] M. Rabinovich, I. Rabinovich, R. Rajaraman, and A. Aggarwal, "A dynamic object replication and migration protocol for an Internet hosting service," in *Proc. 19th IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Austin, TX, USA, May 1999, pp. 101–113.

[34] S. Zaman and D. Grosu, "A distributed algorithm for the replica placement problem," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 9, pp. 1455–1468, Sep. 2011.

[35] U. Lee, I. Rimac, and V. Hilt, "Greening the Internet with content-centric networking," in *Proc. 1st Int. Conf. Energy Efficient Comput. Netw. (e-Energy)*, Passau, Germany, Apr. 2010, pp. 179–182.

[36] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms," in *Proc. USENIX Symp. Internet Technol. Syst.*, Monterey, CA, USA, Dec. 1997, Art. no. 18.

[37] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of Web server replicas," in *Proc. 20th IEEE Conf. Comput. Commun. (INFOCOM)*, Anchorage, AK, USA, Apr. 2001, pp. 1587–1596.

[38] M. Charikar and S. Guha, "Improved combinatorial algorithms for the facility location and k-median problems," in *Proc. 40th IEEE Annu. Symp. Found. Comput. Sci. (FOCS)*, New York, NY, USA, Oct. 1999, pp. 378–388.

[39] S. U. Khan and I. Ahmad, "Comparison and analysis of ten static heuristics-based Internet data replication techniques," *J. Parallel Distrib. Comput.*, vol. 68, no. 2, pp. 113–136, Feb. 2008.

[40] J. Li *et al.*, "Popularity-driven coordinated caching in named data networking," in *Proc. 8th ACM/IEEE Symp. Architect. Netw. Commun. Syst. (ANCS)*, Austin, TX, USA, Oct. 2012, pp. 15–26.

[41] V. Sourlas, P. Flegkas, L. Gkatzikis, and L. Tassiulas, "Autonomic cache management in information-centric networks," in *Proc. 13th IEEE Netw. Oper. Manag. Symp. (NOMS)*, Maui, HI, USA, Apr. 2012, pp. 121–129.

[42] V. Sourlas, P. Flegkas, G. Paschos, D. Katsaros, and L. Tassiulas, "Storage planning and replica assignment in content-centric publish/subscribe networks," *Comput. Netw.*, vol. 55, no. 18, pp. 4021–4032, Dec. 2011.

[43] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. 29th IEEE Conf. Comput. Commun. (INFOCOM)*, San Diego, CA, USA, Mar. 2010, pp. 1–9.

[44] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical Internet object cache," in *Proc. USENIX Annu. Tech. Conf. (ATEC)*, Berkeley, CA, USA, Jan. 1996, Art. no. 13.

[45] S. Jamin, C. Jin, A. R. Kurc, D. Raz, and Y. Shavitt, "Constrained mirror placement on the Internet," in *Proc. 20th IEEE Conf. Comput. Commun. (INFOCOM)*, Anchorage, AK, USA, Apr. 2001, pp. 31–40.

[46] J. Wang, "A survey of Web caching schemes for the Internet," *ACM SIGCOMM Comput. Commun. Rev. Mag.*, vol. 29, no. 5, pp. 36–46, Oct. 1999.

[47] B. Ahlgren *et al.*, "Design considerations for a network of information," in *Proc. 4th ACM Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, Madrid, Spain, Dec. 2008, Art. no. 66.

[48] J. Kangasharju, J. Roberts, and K. W. Ross, "Object replication strategies in content distribution networks," *Comput. Commun.*, vol. 25, no. 4, pp. 376–383, Mar. 2002.

[49] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby, "On the optimal placement of Web proxies in the Internet," in *Proc. 8th IEEE Conf. Comput. Commun. (INFOCOM)*, New York, NY, USA, Mar. 1999, pp. 1282–1290.

[50] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, Feb. 2004.

[51] *Caida Dataset*. Accessed on Feb. 15, 2015. [Online]. Available: http://www.caida.org/research/topology/datasets

[52] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *Proc. 15th Conf. Comput. Commun. (INFOCOM)*, San Francisco, CA, USA, Mar. 1996, pp. 594–602.

[53] Z. Li and G. Simon, "Time-shifted TV in content centric networks: The case for cooperative in-network caching," in *Proc. 53rd IEEE Int. Conf. Commun. (ICC)*, Kyoto, Japan, Jun. 2011, pp. 1–6.

[54] Z. Ming, M. Xu, and D. Wang, "Age-based cooperative caching in information-centric networks," in *Proc. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Orlando, FL, USA, Mar. 2012, pp. 268–273.

[55] J. M. Wang, J. Zhang, and B. Bensaou, "Intra-AS cooperative caching for content-centric networks," in *Proc. 3rd ACM SIGCOMM Workshop Inf. Centric Netw. (ICN)*, Hong Kong, Aug. 2013, pp. 61–66.

[56] D. Rossi and G. Rossini, "Caching performance of content centric networks under multi-path routing (and more)," Dept. Comput. Sci. Netw., Telecom ParisTech Ecole, Paris, France, Tech. Rep. 1, Nov. 2011.

[57] G. Rossini and D. Rossi, "A dive into the caching performance of content centric networking," in *Proc. 17th Int. Workshop Comput. Aided Model. Design Commun. Links Netw. (CAMAD)*, Barcelona, Spain, Sep. 2012, pp. 105–109.

[58] K. Cho *et al.*, "WAVE: Popularity-based and collaborative in-network caching for content-oriented networks," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Orlando, FL, USA, Mar. 2012, pp. 316–321.

[59] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *Proc. 2nd Workshop Inf. Centric Netw. (ICN)*, Helsinki, Finland, 2012, pp. 55–60.

[60] V. Sourlas, P. Flegkas, and L. Tassiulas, "A novel cache aware routing scheme for information-centric networks," *Comput. Netw.*, vol. 59, pp. 44–61, Feb. 2014.

[61] F. M. Al-Turjman, A. E. Al-Fagih, and H. S. Hassanein, "A value-based cache replacement approach for information-centric networks," in *Proc. 38th Conf. Local Comput. Netw. Workshop (LCN)*, Sydney, NSW, Australia, Oct. 2013, pp. 874–881.

[62] J. M. Wang and B. Bensaou, "Progressive caching in CCN," in *Proc. 31st IEEE Glob. Commun. Conf. (GLOBECOM)*, Anaheim, CA, USA, Dec. 2012, pp. 2727–2732.

[63] A. Detti, N. B. Melazzi, S. Salsano, and M. Pomposini, "Conet: A content centric inter-networking architecture," in *Proc. 1st ACM SIGCOMM Workshop Inf. Centric Netw. (ICN)*, Toronto, ON, Canada, Aug. 2011, pp. 50–55.

[64] M. D'Ambrosio, C. Dannewitz, H. Karl, and V. Vercellone, "MDHT: A hierarchical name resolution service for information-centric networks," in *Proc. 1st ACM SIGCOMM Workshop Inf. Centric Netw. (ICN)*, Toronto, ON, Canada, Aug. 2011, pp. 7–12.

[65] P. Krishnan, D. Raz, and Y. Shavitt, "The cache location problem," *IEEE/ACM Trans. Netw.*, vol. 8, no. 5, pp. 568–582, Oct. 2000.

[66] S. Bhattacharjee, K. L. Calvert, and E. W. Zegura, "Self-organizing wide-area network caches," in *Proc. 17th IEEE Conf. Comput. Commun. (INFOCOM)*, San Francisco, CA, USA, Mar./Apr. 1998, pp. 600–608.

[67] 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, no. 3, pp. 281–293, Jun. 2000.

[68] A. Wolman *et al.*, "On the scale and performance of cooperative Web proxy caching," in *Proc. 17th ACM Symp. Oper. Syst. Principles (SOSP)*, Charleston, SC, USA, Dec. 1999, pp. 16–31.

[69] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[70] B. Bollobás, O. Riordan, J. Spencer, and G. Tusnády, "The degree sequence of a scale-free random graph process," *Random Struct. Algorithm.*, vol. 18, no. 3, pp. 279–290, May 2001.

[71] S. N. Dorogovtsev, J. F. F. Mendes, and A. N. Samukhin, "Structure of growing networks with preferential linking," *Phys. Rev. Lett.*, vol. 85, no. 21, pp. 4633–4636, Nov. 2000.

[72] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Mod. Phys.*, vol. 74, no. 1, Jan. 2002, Art. no. 47.

[73] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache 'less for more' in information-centric networks," in *Proc. 11th Conf. Int. Fed. Inf. Process. Netw. (IFIP)*, May 2012, pp. 27–40.

[74] V. Almeida, A. Bestavros, M. Crovella, and A. De Oliveira, "Characterizing reference locality in the WWW," in *Proc. 4th IEEE Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Miami Beach, FL, USA, Dec. 1996, pp. 92–103.

[75] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. 18th IEEE Conf. Comput. Commun. (INFOCOM)*, New York, NY, USA, Mar. 1999, pp. 126–134.

[76] K. P. Gummadi *et al.*, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proc. 9th ACM Symp. Oper. Syst. Principles (SOSP)*, Oct. 2003, pp. 314–329.

[77] G. Dán and N. Carlsson, "Power-law revisited: Large scale measurement study of P2P content popularity," in *Proc. 9th Int. Workshop Peer-to-Peer Syst. (IPTPS)*, San Jose, CA, USA, Apr. 2010, Art. no. 12.

[78] C. Imbrenda, L. Muscariello, and D. Rossi, "Analyzing cacheable traffic in ISP access networks for micro CDN applications via content-centric networking," in *Proc. 1st ACM Int. Conf. Inf. Centric Netw. (ICN)*, Paris, France, Sep. 2014, pp. 57–66.

[79] V. Sourlas, G. S. Paschos, P. Flegkas, and L. Tassiulas, "Caching in content-based publish/subscribe systems," in *Proc. 28th IEEE Glob. Telecommun. Conf. (GLOBECOM)*, Honolulu, HI, USA, Nov. 2009, pp. 1–6.

[80] J. Ardelius, B. Grönvall, L. Westberg, and Å. Arvidsson, "On the effects of caching in access aggregation networks," in *Proc. 2nd Workshop Inf. Centric Netw. (ICN)*, Helsinki, Finland, Aug. 2012, pp. 67–72.

[81] P. R. Jelenković and X. Kang, "Characterizing the miss sequence of the LRU cache," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 2, pp. 119–121, Sep. 2008.

[82] E. J. Rosensweig, J. Kurose, and D. Towsley, "Approximate models for general cache networks," in *Proc. 29th IEEE Conf. Comput. Commun. (INFOCOM)*, San Diego, CA, USA, Mar. 2010, pp. 1–9.

[83] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Modeling data transfer in content-centric networking," in *Proc. 23rd Int. Teletraffic Congr. (ITC)*, San Francisco, CA, USA, Sep. 2011, pp. 111–118.

[84] L. Muscariello, G. Carofiglio, and M. Gallo, "Bandwidth and storage sharing performance in information centric networking," in *Proc. 1st Workshop Inf. Centric Netw. (ICN)*, Toronto, ON, Canada, Aug. 2011, pp. 26–31.

[85] A. Dabirmoghaddam, M. M. Barijough, and J. Garcia-Luna-Aceves, "Understanding optimal caching and opportunistic caching at 'the edge' of information-centric networks," in *Proc. 1st ACM Int. Conf. Inf. Centric Netw. (ICN)*, Paris, France, Sep. 2014, pp. 47–56.

[86] L. Cherkasova and M. Gupta, "Analysis of enterprise media server workloads: Access patterns, locality, content evolution, and rates of change," *IEEE/ACM Trans. Netw.*, vol. 12, no. 5, pp. 781–794, Oct. 2004.
[87] R. Fonseca, V. Almeida, M. Crovella, and B. Abrahao, "On the intrinsic locality properties of Web reference streams," in *Proc. 22nd IEEE Conf. Comput. Commun. (INFOCOM)*, San Francisco, CA, USA, Mar. 2003, pp. 448–458.
[88] G. Rossini and D. Rossi, "Coupling caching and forwarding: Benefits, analysis, and implementation," in *Proc. 1st ACM Conf. Inf. Centric Netw. (ICN)*, Paris, France, Sep. 2014, pp. 127–136.
[89] G. D. M. B. Domingues, E. A. D. S. e Silva, R. M. M. Leão, and D. S. Mensché, "Enabling information centric networks through opportunistic search, routing and caching," *CORR*, abs/1310.8258, Oct. 2013.
[90] R. Chiocchetti, D. Perino, G. Carofiglio, D. Rossi, and G. Rossini, "INFORM: A dynamic interest forwarding mechanism for information centric networking," in *Proc. 3rd Workshop Inf. Centric Netw. (ICN)*, Hong Kong, Aug. 2013, pp. 9–14.
[91] C. J. C. H. Watkins and P. Dayan, "Technical note: Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, May 1992.
[92] R. Chiocchetti, D. Rossi, G. Rossini, G. Carofiglio, and D. Perino, "Exploit the known or explore the unknown?: Hamlet-like doubts in ICN," in *Proc. 2nd Workshop Inf. Centric Netw. (ICN)*, Helsinki, Finland, Aug. 2012, pp. 7–12.
[93] C. Yi *et al.*, "A case for stateful forwarding plane," *Comput. Commun.*, vol. 36, no. 7, pp. 779–791, 2013.
[94] S. K. Fayazbakhsh *et al.*, "Less pain, most of the gain: Incrementally deployable ICN," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, Hong Kong, Aug. 2013, pp. 147–158.
[95] M. Badov, A. Seetharam, J. Kurose, V. Firoiu, and S. Nanda, "Congestion-aware caching and search in information-centric networks," in *Proc. 1st ACM Int. Conf. Inf. Centric Netw. (ICN)*, Paris, France, Sep. 2014, pp. 37–46.
[96] S. Eum, K. Nakauchi, M. Murata, Y. Shoji, and N. Nishinaga, "CATT: Potential based routing with content caching for ICN," in *Proc. 2nd Workshop Inf. Centric Netw. (ICN)*, Helsinki, Finland, Aug. 2012, pp. 49–54.
[97] Y. Zhu, M. Chen, and A. Nakao, "CONIC: Content-oriented network with indexed caching," in *Proc. 29th IEEE Conf. Comput. Commun. (INFOCOM)*, San Diego, CA, USA, Mar. 2010, pp. 1–6.
[98] M. Lee, K. Cho, K. Park, T. Kwon, and Y. Choi, "SCAN: Scalable content routing for content-aware networking," in *Proc. 53rd IEEE Int. Conf. Commun. (ICC)*, Kyoto, Japan, Jun. 2011, pp. 1–5.
[99] N. Laoutaris, H. Che, and I. Stavrakakis, "The LCD interconnection of LRU caches and its analysis," *Perform. Eval.*, vol. 63, no. 7, pp. 609–634, 2006.
[100] H. Che, Y. Tung, and Z. Wang, "Hierarchical Web caching systems: Modeling, design and experimental results," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 7, pp. 1305–1314, Sep. 2002.
[101] T. M. Wong and J. Wilkes, "My cache or yours? Making storage more exclusive," in *Proc. USENIX Annu. Tech. Conf. (ATEC)*, Monterey, CA, USA, Jun. 2002, pp. 161–175.
[102] C. Bernardini, T. Silverston, and O. Festor, "MPC: Popularity-based caching strategy for content centric networks," in *Proc. 55th IEEE Int. Conf. Commun. (ICC)*, Budapest, Hungary, Jun. 2013, pp. 3619–3623.
[103] I. Psaras, W. K. Chai, and G. Pavlou, "In-network cache management and resource allocation for information-centric networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 2920–2931, Nov. 2014.
[104] Y. Wang, M. Xu, and Z. Feng, "Hop-based probabilistic caching for information-centric networks," in *Proc. 32nd IEEE Conf. Glob. Commun. (GLOBECOM)*, Atlanta, GA, USA, Dec. 2013, pp. 2102–2107.
[105] A. Ioannou and S. Weber, "Towards on-path caching alternatives in information-centric networks," in *Proc. 39th IEEE Conf. Local Comput. Netw. (LCN)*, Edmonton, AB, Canada, Sep. 2014, pp. 362–365.
[106] X. Hu and J. Gong, "Opportunistic on-path caching for named data networking," *IEICE Trans. Commun.*, vol. 97-B, no. 11, pp. 2360–2367, Nov. 2014.
[107] W. Wang *et al.*, "CRCache: Exploiting the correlation between content popularity and network topology information for ICN caching," in *Proc. 56th IEEE Int. Conf. Commun. (ICC)*, Sydney, NSW, Australia, Jun. 2014, pp. 3191–3196.
[108] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, Jul. 1990.
[109] M. Busari and C. Williamson, "ProWGen: A synthetic workload generation tool for simulation evaluation of Web proxy caches," *Comput. Netw.*, vol. 38, no. 6, pp. 779–794, Apr. 2002.
[110] G. Rossini and D. Rossi, "On sizing CCN content stores by exploiting topological information," in *Proc. IEEE Conf. Comput. Commun. Workshops*, Orlando, FL, USA, Mar. 2012, pp. 280–285.
[111] A. Anand, V. Sekar, and A. Akella, "SmartRE: An architecture for coordinated network-wide redundancy elimination," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, Barcelona, Spain, Aug. 2009, pp. 87–98.
[112] A. Mahanti, D. Eager, and C. Williamson, "Temporal locality and its impact on Web proxy cache performance," *Perform. Eval.*, vol. 42, nos. 2–3, pp. 187–203, Oct. 2000.
[113] T. Janaszka, D. Bursztynowski, and M. Dzida, "On popularity-based load balancing in content networks," in *Proc. 24th Int. Teletraffic Congr. (ITC)*, Kraków, Poland, Sep. 2012, pp. 1–8.
[114] M. Xie, I. Widjaja, and H. Wang, "Enhancing cache robustness for content-centric networking," in *Proc. 31st IEEE Annu. Int. Conf. Comput. Commun. (INFOCOM)*, Orlando, FL, USA, Mar. 2012, pp. 2426–2434.
[115] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee, "Measurement and analysis of online social networks," in *Proc. 7th ACM Internet Measur. Conf. (IMC)*, San Diego, CA, USA, Oct. 2007, pp. 29–42.
[116] E. Yeh, T. Ho, Y. Cui, M. Burd, R. Liu, and D. Leong, "VIP: A framework for joint dynamic forwarding and caching in named data networks," in *Proc. 1st ACM Int. Conf. Inf. Centric Netw. (ICN)*, Paris, France, Sep. 2014, pp. 117–126.
[117] A. Ioannou and S. Weber, "Information-centric networking: A thorough evaluation of popularity-based probabilistic on-path caching," Dept. Comput. Sci. Statist., Trinity College Dublin, Dublin, Ireland, Tech. Rep. 1, Dec. 2014.
[118] Y. Sun *et al.*, "Trace-driven analysis of ICN caching algorithms on video-on-demand workloads," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, Sydney, NSW, Australia, Dec. 2014, pp. 363–376.
[119] A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM: NDN simulator for NS-3," Dept. Comput. Sci., Named-Data Networking Project, Tech. Rep. NDN-0005, Oct. 2012.
[120] R. Chiocchetti, D. Rossi, and G. Rossini, "ccnSIM: An highly scalable CCN simulator," in *Proc. 55th IEEE Int. Conf. Commun. (ICC)*, Budapest, Hungary, Jun. 2013, pp. 2309–2314.
[121] C. M. S. Cabral, C. E. Rothenberg, and M. F. Magalhães, "Mini-CCNx: Fast prototyping for named data networking," in *Proc. 3rd Workshop Inf. Centric Netw. (ICN)*, Hong Kong, Sep. 2013, pp. 33–34.
[122] V. Martina, M. Garetto, and E. Leonardi, "A unified approach to the performance analysis of caching systems," in *Proc. 33rd IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Toronto, ON, Canada, Apr. 2014, pp. 2040–2048.
[123] J. Kurose, "Information-centric networking: The evolution from circuits to packets to content," *Comput. Netw.*, vol. 66, pp. 112–120, Jun. 2014.
[124] M. Tortelli, D. Rossi, G. Boggia, and L. A. Grieco, "Pedestrian crossing: The long and winding road toward fair cross-comparison of ICN quality," in *Proc. 10th IEEE Int. Conf. Heterogeneous Netw. Qual. Rel. Security Robust. (QShine)*, Aug. 2014, pp. 142–145.

Andriana Ioannou received the bachelor's degree from the Aristotle University of Thessaloniki, Greece, in 2010. She currently pursuing the Ph.D. degree with the Group of Distributed Systems, School of Computer Science and Statistics, Trinity College, Dublin. Her current research interests lie on the area of information-centric networking.

Stefan Weber received the Ph.D. degree from Trinity College, Dublin, in 2002, where he currently is a Professor with the School of Computer Science and Statistics. His current research interests include information-centric networking, emergent networks such as mobile ad hoc networks, and protocols and architectures for the next generation of the Internet.