A Survey of Caching Mechanisms in Information-Centric Networking

Meng Zhang, Hongbin Luo, and Hongke Zhang

***Abstract*—Information-Centric Networking (ICN) is a novel networking paradigm that attracts increasing research interests in recent years. In-network caching has been viewed as an attractive feature of ICN because it can reduce network traffic, alleviate server bottleneck, and reduce the user access latencies. Because of this, the network community has proposed many in-network caching mechanisms that aim at optimizing various performance metrics such as cache hit ratio and cache hit distance. In this survey, we present a comprehensive overview of the recently proposed in-network caching mechanisms for ICN. For each caching mechanism, we describe it in detail, present examples to illustrate how it works, and analyze its possible benefits and drawbacks. We also compare some typical in-network caching mechanisms through extensive simulations and discuss the remaining research challenges.**

***Index Terms*—Information-Centric Networking, in-network caching, cache cooperation, cache efficiency.**

I. Introduction

In THE PAST few decades, the current Internet has evolved from an academic network into a global cyber infrastructure. It is estimated that the annual network traffic of the Internet will surpass the zettabyte (i.e., 1,000 exabytes) threshold in 2016, and the video traffic will account for 79 percent of all consumer Internet traffic [1]. However, the current Internet architecture was centered on the host-to-host communication model, which makes it inefficient to support content distribution and information sharing. Accordingly, in recent years the networking community is actively designing novel Internet architectures from scratch, aiming at rectifying one or more of the current Internet's drawbacks such as poor security and scalability. As a result, it is widely recognized that Information-Centric Networking (ICN) is a promising solution in meeting the needs of future networks [2].

The history of ICN can be dated back to TRAID [3], [4]. After Data-Oriented Network Architecture (DONA) [5] and Content Centric Networking/Named Data Networking (CCN/NDN) [6], [7] were proposed, ICN has attracted plenty of research interests from the networking community. Many projects for investigating ICN have been funded around the world. For example, the European Union funded the Publish-Subscribe Internet Routing Paradigm (PSIRP) [8] (later in the Publish-Subscribe Internet Technology (PURSUIT) [9]) project, the 4WARD [10] (later in the Scalable and Adaptive Internet Solutions (SAIL) [11]) project, and the Content Mediator Architecture for Content Aware Networks (COMET) [12] project. Similarly, the United States funded the NDN project and the MobilityFirst [13] project.

Different from the current Internet that assigns hosts/interfaces IP addresses and transfers data packets amongst end points by using IP addresses, ICN assigns contents (instead of hosts/interfaces) names that are irrelevant to their locations. In addition, with ICN, users consume contents by sending requests including content names to the network, which then routes the requests to nearby copies of the desired contents by using the content names (instead of IP addresses). This in turn makes it attractive for ICN to use in-network caching since doing this cannot only reduce the latency for users to access contents, but also can reduce network traffic [14], [15]. For example, the trace-driven analysis in [16] demonstrates that in-network caching can help to save three or more hops for 30% of packets, or equivalently 2.02 hops being saved on average, and a considerable proportion of requests (11%) can be retained within the requester's domain. Because of these benefits, the above mentioned ICN architectures (such as NDN and DONA) all use ubiquitous in-network caching, though they have different design objectives and details.

To make best usage of in-network caching, the networking community has proposed many caching mechanisms in recent years. In this paper, we attempt to make an overview of these caching mechanisms. In particular, we make the following contributions:

Firstly, we make an overview of the recently proposed caching mechanisms, focusing on their basic ideas, their benefits and possible drawbacks. Specifically, we classify them into several categories, based on their properties. For example, based on whether a content chunk is cached along its delivery path, in-network caching can be classified into on-path caching and off-path caching. We also define a set of performance measures that quantify the benefits and shortcomings of the various mechanisms.

Secondly, we conduct extensive simulations to compare the performance of several typical caching mechanisms and to investigate the key factors that affect the caching performance, as in the case of heterogeneous caching or cooperative caching.

Manuscript received June 4, 2014; revised January 13, 2015; accepted March 18, 2015. Date of publication April 6, 2015; date of current version August 20, 2015. This work was supported in part by the Canadian NSERC; by the National Natural Science Foundation of China under Grants This work was supported in part by the National Science Foundation under Grant Nos. 61422101, 61271200, 61232017, in part by the National Basic Research Program ("973 Program") of China under Grant No. 2013CB329100, in part by the Ph.D. Programs Foundation of the Ministry of Education of China under Grant No. 20130009110014, in part by "NCET" under Grant No. NCET-12-0767, and in part by the Fundamental Research Funds for the Central Universities under Grant No. 2014JBM011. *(Corresponding author: Hongbin Luo.)*

The authors are with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China (e-mail: mengzhang@bjtu.edu.cn; hbluo@bjtu.edu.cn; hkzhang@bjtu.edu.cn).

Digital Object Identifier 10.1109/COMST.2015.2420097

TABLE I
FREQUENTLY USED ACRONYMS

Short form	Long form
ICN	Information-Centric Networking
DONA	Data-Oriented Network Architecture
CCN/NDN	Content Centric Networking/Named Data Networking
LRU/LFU	Least Recently Used/Least Frequently Used
PIT	Pending Interest Table
FIB	Forwarding Information Base
CS	Content Store
RH	Resolution Handler
LCST/ECST	Local Cache Summary Table/Exchange Cache Summary Table
TSI/TSB	Time Since Inception/Time Since Birth
CRT/CCS	Collaborative Router Table/Collaborative Content Store
FID	Forwarding IDentifier
PBR	Potential Based Routing
CM	Cache Manager

Last but not the least, we point out the main challenges that require further research by the community.

We note that Zhang *et al.* have made a survey on ICN caching mechanisms [17]. However, our paper differs significantly from that work in several ways. Firstly, we focus on the most recently proposed caching mechanisms for ICN and describe every approach in detail by using examples to illustrate their fundamental operations. By contrast, the work in [17] briefly introduces only some of the caching mechanisms. Secondly, this paper proposes a categorization criteria for ICN caching mechanisms. Thirdly, we also conduct extensive simulations to compare some typical mechanisms. Finally, by summarizing the existing caching mechanisms, we elaborate on their characteristics, similarities and differences.

Note that we use "content" and "Data packet" interchangeably to refer to a cacheable content chunk, which is not necessarily of similar size to the traditional IP packet. In fact, the size of cacheable content chunks is still an open issue and is out the scope of this paper. In addition, Table I summarizes the acronyms used in this paper for better readability.

The rest of this paper is organized as follows. In Section II, we describe two typical ICN architectures (i.e., NDN and DONA) to lay the basis of this paper. In Section III, we list the performance measures for evaluating caching mechanisms and present the criteria used for classifying the in-network caching mechanisms. In Section IV, we give a survey of different in-network caching mechanisms. In Section V, we present the simulation results of the typical caching mechanisms and compare their caching performance. Finally, we discuss the open problems and conclude the paper in Sections VI and VII, respectively.

II. INFORMATION-CENTRIC NETWORKING ARCHITECTURES

In this section, we briefly introduce two different ICN architectures (i.e., NDN and DONA) to lay the foundation for the in-depth discussions of this paper. We refer interested readers to [2] for a comprehensive survey of ICN architectures. It is worth noting that all ICN architectures have considered in-network caching as a basic feature.

A. NDN

NDN (also known as CCN [18]) is aimed at replacing "where" with "what" in the thin waist of the Internet's hourglass architecture. Unlike the current Internet that assigns IP addresses to network hosts/interfaces, NDN assigns names to content. In particular, each content chunk is assigned with a hierarchically structured name that is similar to the current *Uniform Resource Locators* (URLs). Each component of the name can be anything, such as a string or a hash value. In addition, NDN prefers human-readable names. For example, it can be www.example.com/logo.jpg (EX for short), where the slash denotes the delimiter between different components of a name. This naming style makes it possible for users to remember the names, and, to some extent, to establish the mapping between a content name and what the user wants.

Content retrieval in NDN relies on two types of packets: Interest packets and Data packets. Users interested in a certain content chunk send out Interest packets into the network, which replies with the Data packets corresponding to the Interest packets. To achieve this, content routers in NDN propagate the reachability information of content names by using such routing protocols as OSPF and BGP, in a way like the current Internet propagates the reachability information of IP-prefixes. In addition, a content router maintains a *Pending Interest Table* (PIT), a *Forwarding Information Base* (FIB), and a *Content Store* (CS). The FIB which stores the reachability information of content names is used to forward Interest packets toward the content source(s). The PIT of a content router records the interface(s) that Interest packets arrive at the content router, so that the returned Data packets can be correctly sent to its original requester(s). Note that, if a content router receives

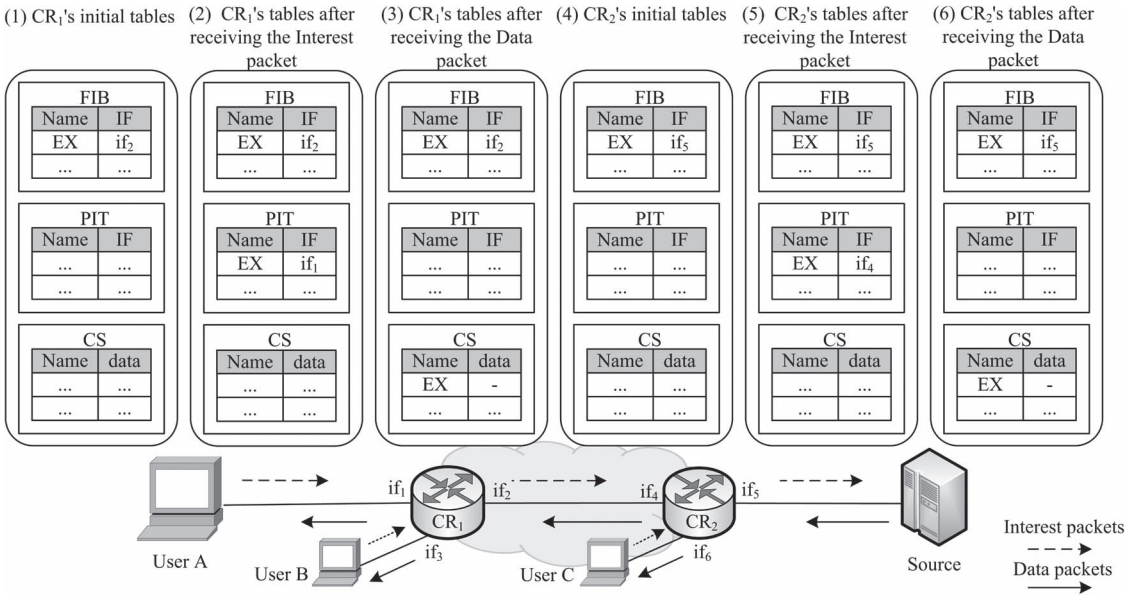


Fig. 1. Illustration for content retrieval in NDN. IF stands for Interface. CR stands for Content Router.

interests for the same content chunk from multiple interfaces, it should record all these interfaces in its PIT. The CS is used to cache some recently forwarded content chunks.

When a NDN content router receives an Interest packet for a content chunk named by a unique content name, it directly responds the Interest packet with the corresponding Data packet if it caches the copy in its CS. Otherwise, it inquires its PIT. If the content router finds an entry for the content name in its PIT, it records the interface from which the Interest packet arrives at the content router. If the content router cannot find an entry for the content name in its PIT, it forwards the Interest packet toward the content source after querying its FIB.

When a content router receives a Data packet, it firstly caches the Data packet into its CS. It then queries its PIT and forwards the Data packet to the next hop(s) by sending out the Data packet to the interface(s) recorded in the PIT entry for the corresponding content name. After that, the content router deletes the entry for the content name from its PIT.

We now illustrate how to retrieve contents in NDN, with the help of Fig. 1. We assume that the content source provides a desired content named EX and the reachability information of this content name is propagated by using a routing protocol such as OSPF or BGP. As a result, the FIB of CR_1 and that of CR_2 maintain an entry for the content name, as illustrated by (1) and (4) in Fig. 1, respectively. To obtain the desired content, a given user A firstly sends out an Interest packet containing the content name EX to its first-hop content router CR_1. When CR_1 receives the Interest packet, it cannot find a cached copy of the desired content in its CS. Accordingly, it queries its PIT but cannot find an entry for the content name. As a result, CR_1 stores an entry for the content name in its PIT, as illustrated by (2) in Fig. 1. After that, CR_1 queries its FIB and then forwards the Interest packet to CR_2 through the interface if_2. Similarly, when CR_2 receives the Interest packet, it cannot find a cached copy of the desired content in its CS. In addition, since CR_2 cannot find an entry for the content name in its PIT, it also adds a new entry for the content name in its PIT, as illustrated by (5) in Fig. 1. CR_2 then queries its FIB and forwards the Interest packet to the content source.

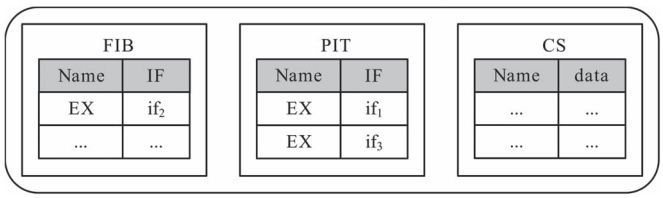


Fig. 2. CR_1's tables after CR_1 receives the Interest packet from *B* before the corresponding Data packet arrives. IF stands for Interface.

When the content source receives the Interest packet, it sends the corresponding Data packet to CR_2. When CR_2 receives the Data packet, it firstly stores the Data packet into its CS. After that, it queries its PIT and finds that it should send out the Data packet from the interface if_4. After sending out the Data packet through if_4, CR_2 deletes the entry for the content name corresponding to the Data packet from its PIT, as illustrated by (6) in Fig. 1. Similarly, when CR_1 receives the Data packet, it stores the Data packet into its CS, sends out the Data packet through if_1, and deletes the entry for the content name from its PIT, as illustrated by (3) in Fig. 1.

Note that, if CR_1 receives an Interest packet for the same content from B through if_3 before it receives the corresponding Data packet, it adds an entry for the content name into its PIT, as illustrated in Fig. 2. Later when CR_1 receives the corresponding Data packet, it should send out the Data packet through both interfaces if_1 and if_3. After that, it deletes the corresponding entries in its PIT. The resulting tables of CR_1 are presented in Fig. 1 (3).

Now assume that another user C attaching to CR_2 through if_6 also wants to obtain the content named by EX. For this purpose, C sends an Interest packet to CR_2. When CR_2

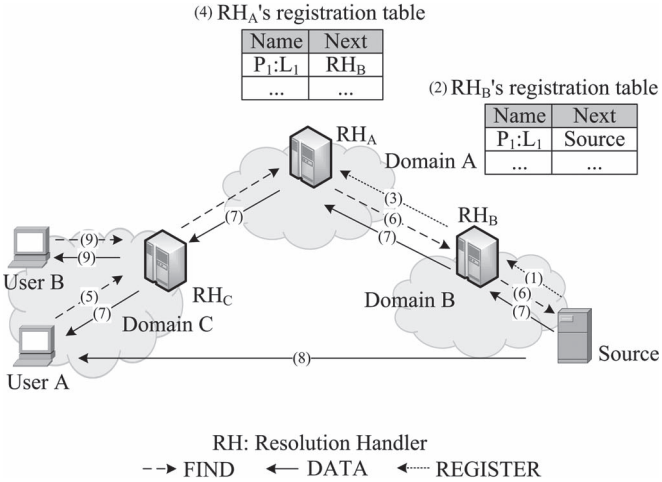


Fig. 3. Illustration for content registration and retrieval in DONA.

receives the Interest packet, it can find a cached copy of the desired content in its CS. Therefore, CR_2 directly sends the desired Data packet to the user, without obtaining the desired Data packet from the content source.

From the above descriptions, it is clear that content routers in NDN are required to be capable of caching Data packets. In addition, every content router along the path caches the Data packet. This is called "*Caching Everything Everywhere (CEE)*". As a result, the same Data packet may be cached at multiple content routers. It is possible that there is no free storage space in a content router's CS when the content router receives a Data packet. In this case, the content router may evict some Data packets stored in its cache, based on some cache replacement algorithms such as LRU [19]–[23], LFU (*Least Frequently Used*) [15], MRU (*Most Recently Used*) and MFU (*Most Frequently Used*) [24], [25]. It is important to note that content routers should be able to cache content chunks at line-speed in ICN [26]. Since there is a comprehensive survey on cache replacement strategies, we refer interested readers to the paper [27]. Instead, we strictly focus on in-network caching mechanisms.

B. DONA

DONA is another ICN architecture that attracts plenty of research interests. For persistence and uniqueness of contents, DONA uses self-certifying names (instead of the traditional DNS-based names). In particular, the naming format can be expressed as "$P : L$", where P denotes the hash of the publisher's public key and L represents the unique label assigned to a content by the publisher.

DONA employs a tree-based routing scheme, as illustrated in Fig. 3. In particular, DONA assumes that a future Internet is still comprised by domains that have the provider/customer/peer relationship. In addition, a domain in DONA maintains a *Resolution Handler* (RH) to implement content registration and retrieval functions. Furthermore, the RHs share the same provider/customer/peer relationship with the domains that they locate at.

DONA uses three basic primitives: REGISTER, FIND and DATA. When a content source is authorized to serve the content, it sends a REGISTER message to its local RH, as illustrated by (1) in Fig. 3. When the RH receives the REGISTER message from the content source, it adds an entry for the content name into its registration table, as illustrated by (2) in Fig. 3. The entry records the content name and the IP address of a remote RH (or content source) from which the RH receives the REGISTER message. For example, the entry for the content name $P_1 : L_1$ in RH_B's registration table points to the content source, as illustrated by (2) in Fig. 3. The RH then forwards the REGISTER message to its parents and peers based on its local policy, as illustrated by (3) in Fig. 3. Similarly, the entry in RH_A's registration table points to RH_B, as illustrated by (4) in Fig. 3. This process terminates when the REGISTER message is registered to the Tier-1 *Internet Service Providers* (ISPs). We note that DONA only registers the availability information of these contents that are published by content sources.

When a user wants to obtain a content, it sends out a FIND packet to its local RH, as illustrated by (5) in Fig. 3. When a RH receives a FIND packet, it firstly looks up its registration table. If there is an entry for the content name in its registration table, the RH forwards the FIND packet to the corresponding next hop. Otherwise, it forwards the FIND packet to one of its parent RHs, based on its local policy. This way, the FIND packet will either be forwarded to the content source, as illustrated by (6) in Fig. 3, or be discarded by the RH of a Tier-1 ISP. When the content source receives the FIND packet, it sends the corresponding DATA packets to the requester through the reverse path of the FIND packet, as illustrated by (7) in Fig. 3, or through a direct route as illustrated by (8) in Fig. 3.

DONA also supports in-network caching. To achieve this, the RHs need to be cache-capable and the DATA packets should be forwarded from a server to the requester through the reverse path of the corresponding FIND packets, as illustrated by (7) in Fig. 3. Once a RH caches a DATA packet, it can response subsequent FIND packets by directly sending the cached DATA packets to the requester, as illustrated by (9) in Fig. 3.

III. Performance Measures and Categories of Caching Mechanisms

In this section, we describe the performance measures for evaluating caching mechanisms and present the criteria used for classifying them.

A. Performance Measures

1) Cache Hit Ratio: A cache hit means that a request for a content chunk can be satisfied by a cached copy. On the other hand, if a request cannot be satisfied by a cached copy, we say that a cache miss occurs. Accordingly, the cache hit ratio is defined as the ratio of the number of cache hits to the sum of the number of cache hits and cache misses. Intuitively, a higher cache hit ratio can reduce both the service access latency and the server load.

2) Cache Hit Distance: Considering a service request r for a given content chunk, let $H(r)$ be the distance from the requester

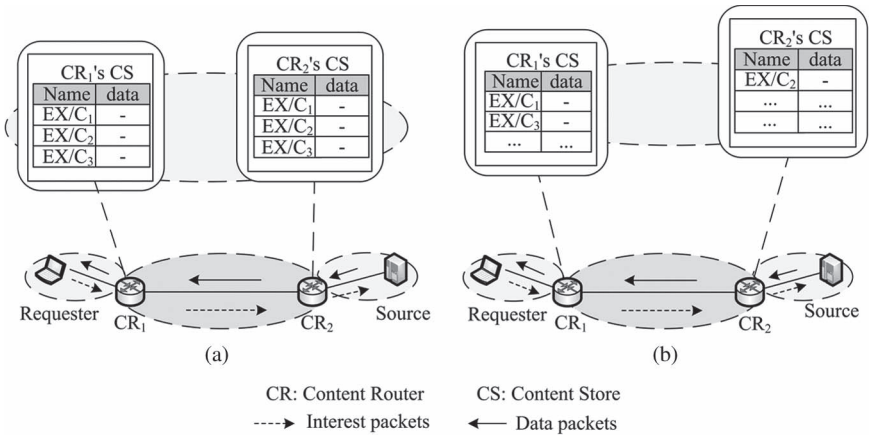


Fig. 4. Illustration of homogeneous caching and heterogeneous caching. (a) Homogeneous caching; (b) heterogeneous caching.

to the content router that answers the service request with the corresponding data packet. The hit distance is then defined as the average of $H(r)$ over all service requests. It is clear that a shorter cache hit distance is preferred since it means better user experience.

3) Operational Cost: Many caching mechanisms advertise the availability of a cached copy to the network and use a *Name Resolution Service* (NRS) to route service requests to the cached copy. Unlike the content source that permanently stores a piece of content once it is authorized to serve the content, content routers may update its cached contents due to the limitation of cache space. Unfortunately, every time a content router evicts a cached content, it needs to unregister the cached content such that future service requests are not routed to the content router. This in turn leads to a large amount of advertisement messages or unregister messages. We define the operational cost as the total number of advertisement messages and unregister messages. Therefore, it is desirable that the operational cost is as low as possible, so that a caching mechanism can be applied to large scale networks.

From the above descriptions, one can see that a caching mechanism that has high hit ratio, low hit distance and low cost is desirable. While existing caching mechanisms are designed for optimizing one or more of the above described measures, they can be divided into multiple categories. Below we present the criteria used for categorization.

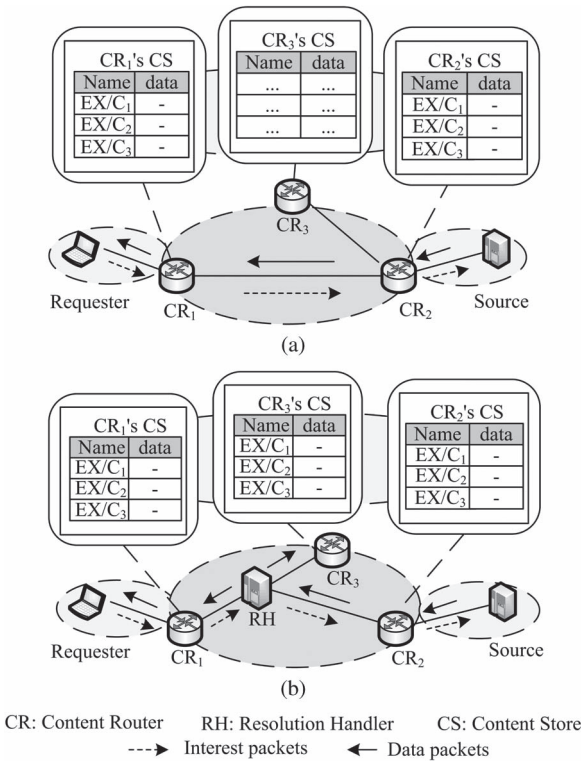


Fig. 5. Illustration of on-path caching and off-path caching. (a) On-path caching; (b) off-path caching.

B. Categorization Criteria

Based on their characteristics, we divide the existing caching mechanisms into multiple categories. While it is a tough job to find appropriate criteria, we classify them into the following categories.

1) Homogeneous Caching VS Heterogeneous Caching: In homogeneous caching, content routers cache the Data packets passed by, and embrace the same cache provision. For example, in Fig. 4(a), content routers (e.g., CR_1 and CR_2) which have a same cache provision, and both cache the content chunks (e.g., EX/C_1, EX/C_2 and EX/C_3. On the other hand, in heterogeneous caching, not all content routers along the downloading path cache the Data packets. For example, in Fig. 4(b), when the requester obtains the three content chunks, CR_1 caches EX/C_1 and EX/C_3, but CR_2 caches EX/C_2. Meanwhile, heterogeneous cache provision is another way to achieve heterogeneous caching. In heterogeneous cache provision, each content router in the network has the different cache size. For instance, the "important" content routers (e.g., the heavily loaded content routers) have more cache size and are therefore able to store more Data packets.

2) Cooperative Caching VS Non-Cooperative Caching: Depending on whether or not the content routers cooperate with each other, caching mechanisms can be classified into cooperative caching and non-cooperative caching. In non-cooperative caching, content routers in a network make caching decisions independently and do not advertise the information of a content

TABLE II
CATEGORIES OF EXISTING CACHING MECHANISMS

Category	Caching Mechanisms
Homogeneous Caching	CEE, Breadcrumbs, Intra-AS Co, ICC;
Heterogeneous Caching	Prob(p), Probcache, CBC, LCD, MCD, WAVE, CLS, CINC, CPHR, LCC, CATT, HSS;
Non-cooperative Caching	CEE, Prob(p), HSS;
Implicit Cooperation	Breadcrumbs, Probcache, CBC, LCD, MCD, WAVE, CLS, CINC, CPHR;
Explicit Cooperation	ICC, Intra-AS Co, LCC, CATT, DCM;
On-path Caching	CEE, Breadcrumbs, Intra-AS Co, Prob(p), Probcache, CBC, LCD, MCD, WAVE, CLS, CPHR, LCC, CATT, HSS;
Off-path Caching	CINC, ICC, DCM.

TABLE III
SUMMARY OF THE CACHING MECHANISMS IN ICN

Short Form	Full Name	References	Content Replication	Cache Retrieval
CEE	Caching Everything Everywhere	[18]	caching everything	locally available
Prob(p)	caching with Probability	[14]	probability caching	locally available
Breadcrumbs	Breadcrumbs	[28]	caching everything	available along the path
Intra-AS Co	Intra-AS Cache cooperation	[29]	caching everything	available between neighbors (one-hop)
ICC	Incentive-Compatible Caching	[30]	caching everything	available between peering domains
Probcache	Probcache	[31], [32]	probability caching	locally available
CBC	Centrality-Based Caching	[33]	caching at the "important" router	locally available
LCD	Leave Copy Down	[14]	caching determined by the flag bit	locally available
MCD	Move Copy Down	[14]	caching determined by the flag bit	locally available
WAVE	WAVE	[34]	caching determined by the flag bit	locally available
CLS	implicit coordinate chunk Caching Location and Searching	[35]	caching determined by the flag bit	available along the path
CINC	Cooperative In-Network Caching	[36]	caching at the content router whose label matches the chunk's sequence number	available within a domain
CPHR	CPHR	[15]	caching at the content router whose label matches the hash value of content name	available within a domain
LCC	Locally Cooperating Caching	[37]	caching at the content router whose label matches the hash value of content name	available within a domain
CATT	Cache Aware Target idenTification	[38]	caching determined by the flag bit	available between neighbors (k-hop)
HSS	Heterogeneous Storage Size	[39]	caching everything	locally available
DCM	Distributed Cache Management	[40]	caching determined by the traffic gain	available within a domain

cached by a content router to other content routers. By contrast, in cooperative caching, content routers in a network cooperate with each other so as to cache more content chunks, or content routers establish cache states to make the cached content available for off-path requests. For example, a subsequent request may be responded by a copy not cached along the path from the content source to the requester, which enables the cached content to respond more requests. Both methods are targeted at improving the utilization efficiency of cache resources. Depending on whether there exists the cache state advertisement mechanism, cooperative caching can be further classified into explicit cooperation and implicit cooperation. In implicit cooperation, content routers do not need to deliver its cache states through an additional advertisement mechanism. But extra operations are required to help content routers make caching or forwarding decisions. Conversely, explicit cooperation requires that the content router should advertise its cache state to other content routers after holding the copy of a piece of content.

3) On-Path Caching VS Off-Path Caching: Depending on the place that a content chunk is cached, caching mechanisms can be classified into on-path caching and off-path caching. In on-path caching, a Data packet is cached along its path to the requester, as illustrated in Fig. 5(a). Note that our definition for on-path caching is different from that in [2], where on-path caching is defined as: a Data packet is cached along the path that the corresponding name resolution request (e.g., an Interest packet in NDN) takes. But in off-path caching, the Data packet may or may not be cached at the content routers along that path, as illustrated in Fig. 5(b). While on-path caching is inherent in ICN, the off-path caching is also possible if a centralized topology manager exists (e.g., a RH). For example, in Fig. 5(b), the RH can decide where to cache a Data packet and forward a copy of the Data packet to the chosen content router(s).

Based on the above classifications, we divide existing caching mechanisms into the above described categories, as shown in Table II.

IV. An Overview of Existing Caching Mechanisms

We now make an overview of the in-network caching mechanisms, as shown in Table III, where we also summarize their content replication and cache retrieval strategies.

A. Homogeneous and Non-Cooperative Caching

CEE: CEE is the default caching mechanism in CCN/NDN, and is designed to minimize the upstream bandwidth demand and downstream latency. In CEE, the Data packet corresponding to an Interest packet from a requester is cached at every content router along the path from the node answering the Interest packet to the requester. Accordingly, CEE is a typical homogeneous and non-cooperative caching mechanism, and faces the problem of caching redundancy because every content router holds the copy of a Data packet. In addition, the retrieval strategy requires that each content router queries its CS whenever it receives an Interest packet. If there is no corresponding copy in the local CS, the Interest packet will be forwarded based on the FIB, making the cached content available only for on-path requests.

B. Heterogeneous and Non-Cooperative Caching

Prob(p): Prob(p) is aimed at reducing the caching redundancy and improving the cache efficiency. For this purpose, each content router in Prob(p) caches a Data packet with probability p, and does not cache it with probability $1-p$. The probability p is set by the network administrator in advance. When a content router receives a Data packet, it randomly generates a number between zero and one. If the generated number is smaller than p, the content router caches a copy of the Data packet. Otherwise, the content router forwards the Data packet without caching it. As a result, a Data packet may be cached at some content routers, but may not be cached at other content routers. Because of this, Prob(p) falls into the heterogeneous caching category.

It is worthy of noting that, with Prob(p), the more times that a content chunk passes through a content router, the higher probability it would be cached at this router. Indeed, after a content chunk passes through a content router n times, the probability that the content chunk will be cached at this router is $1-(1-p)^n$.

Since Prob(p) does not cache a same Data packet at every content router along the downloading path from the node answering the corresponding Interest packet to the requester, it reduces the caching redundancy, thus improving the cache efficiency. In addition, as analyzed above, Prob(p) tends to cache popular contents (i.e., the frequently accessed contents), thus improving the cache hit ratio. Note that the caching performance of Prob(p) depends heavily on the probability p. In particular, when the probability p equals to one, Prob(p) is equivalent to CEE.

In summary, both CEE and Prob(p) are non-cooperative since content routers make their caching decisions independently and don't share their cache states. Content routers forward the Interest packets according to its FIB, so the cached content is available only for on-path requests. In addition, from the above descriptions, both CEE and Prob(p) cache a Data packet at the content routers along the delivery path. Accordingly, they fall into the on-path caching category.

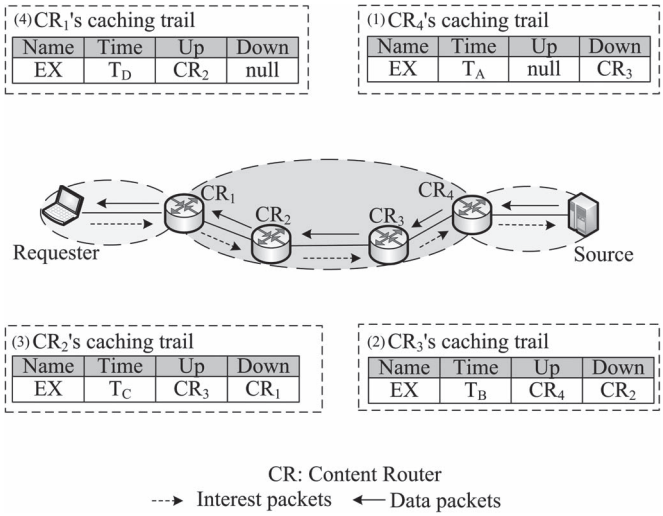


Fig. 6. Illustration of Breadcrumbs operations. Up stands for Upstream Router. Down stands for Downstream Router.

C. Homogeneous and Cooperative Caching

1) Breadcrumbs: Similar to CEE, Breadcrumbs caches a Data packet at every content router along the downloading path. Because of this, it belongs to the category of homogeneous and on-path caching. When a Data packet is cached at a content router, Breadcrumbs creates a data routing history, or called as a caching trail. A caching trail created by a content router (denoted by R_x) is a 4-tuple entry, including the content name, the forwarding time, the identifier of its upstream content router from which the Data packet arrives at R_x, and the identifier of its downstream content router to which the Data packet is forwarded from R_x. Fig. 6 illustrates the basic operations of Breadcrumbs. Firstly, when CR_4 receives a Data packet named by EX, it caches the Data packet in its local CS and sends out the Data packet to the interface corresponding to the entry in the PIT. At the same time, a caching trail is created and maintained at CR_4, as illustrated by (1) in Fig. 6. The values of T_A and CR_3 represent the forwarding time and the identifier of CR_4's downstream content router, respectively. Because the upstream content router of CR_4 is the content source, the *upstream router* field is set to be null. Similarly, when CR_3 receives the Data packet, it caches and forwards the Data packet, then creates a caching trail (EX, T_B, CR_4, CR_2), as illustrated by (2) in Fig. 6. Like CR_4 and CR_3, content routers CR_2 and CR_1 also establish the corresponding caching trails when they forward the Data packet, as illustrated by (3) and (4) in Fig. 6, respectively. Note that the downstream node of CR_1 is the requester, so the *downstream router* field in CR_1's caching trail is set to be null.

The caching trail for a content name at a content router is then used to decide the next hops of Interest packets for the content name. In particular, Breadcrumbs sets a threshold

(denoted by Th_a) for every caching trail. When a content router receives an Interest packet but cannot find a cached copy in its CS, it forwards the Interest packet to its downstream content router (with respect to the content name contained in the Interest packet) if the corresponding content was either cached or refreshed (via cache hit) within the past Th_a period. The rationale behind this operation is: if the LRU or *First in First out* (FIFO) [41] replacement algorithm is used, a recently cached or updated content is more likely to be found in a downstream content router's CS than it could be found in an upstream content router's CS.

If the above condition is met, the Interest packet is forwarded to the downstream content router. Otherwise, it is forwarded to the upstream content router. For example, assume that CR_3 in Fig. 6 receives a locally-unsatisfied Interest packet at the time t but it can find a caching trail for the content name contained in the Interest packet. If $t - T_B$ is smaller than Th_a, CR_3 forwards the Interest packet to the downstream content router CR_2. Otherwise, it forwards the Interest packet to the upstream content router CR_4. Note that if a content router cannot find a caching trail when it receives a locally-unsatisfied Interest packet, it forwards the Interest packet based on its FIB.

From the above descriptions, one can see that Breadcrumbs is a cooperative caching mechanism in which the cached content can be used for off-path requests, because the subsequent Interest packets can be satisfied by content routers that are not along the path from the content source to the requester. However, since content routers do not advertise their cache states, Breadcrumbs uses an implicit cooperation and a content router does not know the cache states of other content routers in the network, which may lead to some drawbacks. Indeed, because of the limited caching capacity, some cached contents are frequently replaced. Accordingly, when a content router forwards a locally-unsatisfied Interest packet to the corresponding downstream content router, the content cached at the downstream content router may be replaced. As a result, it is possible that an Interest packet may be forwarded multiple times between two neighboring content routers, which may lead to either the loss of the Interest packet or longer latency.

To avoid the above shortcomings, the community proposed two explicit cache cooperations, i.e., intra-domain cache cooperation (e.g., Intra-AS Co) and inter-domain cache cooperation (e.g., ICC). We describe them below.

2) Intra-AS Co: Intra-AS Co aims at reducing the caching redundancy by allowing the cached content be visible to neighboring content routers. In particular, every content router in Intra-AS Co maintains two cache summary tables: a *Local Cache Summary Table* (LCST) and an *Exchange Cache Summary Table* (ECST). The LCST of a content router records the information of content chunks currently cached by the content router. On the other hand, the ECST of a content router records the information of contents cached by the router's neighboring content routers. In addition, every content router periodically advertises its local cache summaries to its directly connected neighbors. We describe the operation of Intra-AS Co with the help of the example shown in Fig. 7, where we assume that, at the very beginning, four content chunks EX/C_1, EX/C_2, EX/C_3 and EX/C_4 are cached at CR_1, CR_2, CR_3 and CR_4, respectively. Fig. 7(1) and (2) present the initial cache summaries of CR_1 and CR_2, respectively. Every content router then periodically advertises its local cache summary to its one-hop neighbors. When a content router receives the summaries from its directly connected neighbors, it records the content names and router identifiers into its ECST. For example, after receiving the summary advertised by CR_2, the cache summary of CR_1 is shown by (3) in Fig. 7. When CR_2 receives the cache summaries from CR_1, CR_3, and CR_4, it updates its ECST, as illustrated by (4) in Fig. 7.

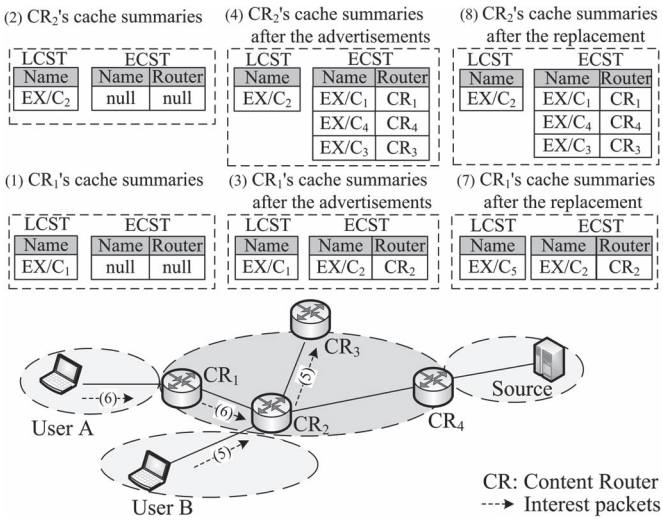


Fig. 7. Illustration of Intra-AS Co operations.

When a content router receives an Interest packet but cannot find a cached copy in its local CS, it queries its ECST to check whether or not one of its directly connected neighbors caches the desired content chunk. If no cached copy is found, the content router then forwards the Interest packet to the next hop according to its FIB. For example, when CR_2 receives an Interest packet for EX/C_3 issued by B, it cannot find a local cached copy in its LCST. Accordingly, CR_2 looks up its ECST and finds that its neighboring router CR_3 has cached a copy of the desired content. Thus CR_2 forwards the Interest packet to CR_3, as illustrated by (5) in Fig. 7. On the other hand, when CR_1 receives an Interest packet for EX/C_3 from A, it cannot find a copy of the desired content in its LCST. In addition, it cannot find an entry for EX/C_3 in its ECST. Accordingly, CR_1 forwards the Interest packet to CR_2 after querying its FIB, as illustrated by (6) in Fig. 7.

Note that a content chunk may be cached at two neighboring nodes, thus leading to caching redundancy. Therefore, an interesting problem is how to choose a set of content routers in a network to cache a given content, so that the number of copies cached in the network is minimized, thus eliminating redundancy. Note that this is the well-known *Minimum Dominating Set* (MDS) problem, which is NP-hard and can be resolved by approximate solutions (e.g., through greedy algorithms [42], [43]). We illustrate the greedy algorithm through an example network shown in Fig. 8, assuming that the content routers cache the same content (e.g., EX/C_1). Let M and N be two sets and they are initially set to be $(CR_1, CR_2, \ldots, CR_9)$

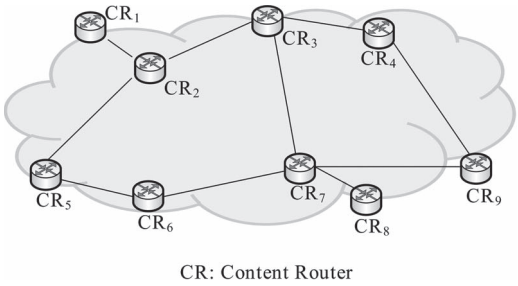


Fig. 8. An example network topology for illustrating the greedy algorithm.

and empty, respectively. For ease of presentation, we define a content router as "covered" by set N if either it is in the set N or one of its directly connected neighbors is in the set N; otherwise, it is called as "uncovered". With this definition, a greedy algorithm may work as follows. At the beginning, all the content routers are uncovered. At each step, the greedy algorithm first selects the content router that has the maximum number of uncovered neighbors and caches the content at this router. After that, the greedy algorithm moves the chosen router from the set M to the set N, and deletes the rest covered router(s) from the set M. The algorithm terminates if the set M is empty. For example, we initially choose CR_7 to cache the content since it has the maximum number of uncovered neighbors (the number = four). Since CR_3, CR_6, CR_7, CR_8 and CR_9 are all covered when CR_7 is chosen to cache the content, we delete CR_3, CR_6, CR_8 and CR_9 from the set M. In addition, the greedy algorithm moves CR_7 from the set M into the set N. Now the set M becomes (CR_1, CR_2, CR_4, and CR_5). Since CR_2 has the maximum number of uncovered neighbors (the number = two, i.e., CR_1 and CR_5), the greedy algorithm chooses CR_2 to cache the content, then deletes CR_1 and CR_5 from the set M, and moves CR_2 from the set M to the set N. As a result, the set M becomes (CR_4). At this time, since the set M has only one content router, the greedy algorithm chooses CR_4 to cache the content and moves CR_4 from the set M to the set N. Now the set M becomes empty and the greedy algorithm terminates. As a result, the set N is set to be (CR_7, CR_2, and CR_4) and the content routers in the set N then hold the copy but the other content routers delete the cached copy. Note that in the case two or more content routers have the same number of uncovered neighbors, the greedy algorithm randomly chooses one of them to hold the copy, while deleting the copies that are cached at these content routers that are not chosen.

One can check that this greedy algorithm works very well, though it may not choose the minimum number of content routers to cache the content. In addition, this greedy algorithm guarantees that, if a content router deletes a cached content because one of its neighbors caches the same content, it is able to find the content in one-hop.

Intra-AS Co uses an explicit cooperation in which content routers explicitly advertise their local cache summaries to their one-hop neighbors so as to make the cached content available for off-path requests. In addition, it is an on-path caching mechanism since the Data packet is only cached along the delivery path. Furthermore, since a Data packet is initially cached at all content routers along the path from the node answering an Interest to the requester, Intra-AS Co is a homogeneous caching mechanism.

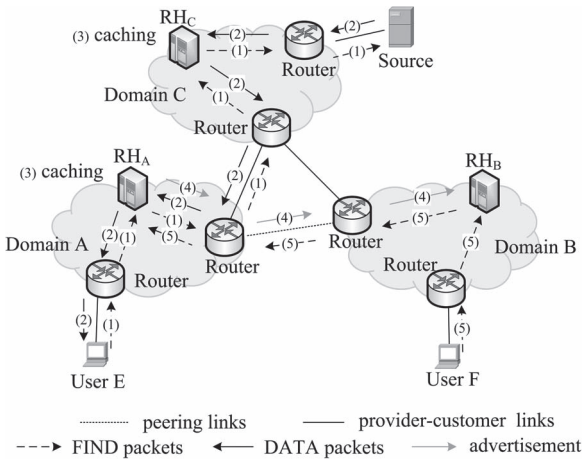


Fig. 9. Illustration of ICC operations.

Note that, because content routers periodically advertise their local cache summaries to neighbors, some entries in a content router's ECST may be outdated. For example, in the case that the content cached at CR_1 (i.e., EX/C_1) is replaced by a new content (e.g., EX/C_5) but CR_1 fails to advertise its new local cache summary to CR_2, the entry for EX/C_1 in CR_2's ECST will be outdated, as illustrated by (7) and (8) in Fig. 7. When an Interest packet for EX/C_1 arrives at CR_2, it will be forwarded to CR_1 which does not cache EX/C_1 anymore. As a result, the Interest packet will either be discarded, or re-forwarded to the content source, resulting in a longer hit distance and access delay.

3) ICC: ICC is a caching approach proposed for DONA and aimed at deploying inter-domain cache cooperation, which is different from the above presented intra-domain cache cooperation. In DONA, RHs could be equipped with caching capabilities of caching every incoming DATA packet. However, RHs in DONA do not advertise the availability information of their cached contents to other RHs. To improve the availability of cached content, ICC proposes that RHs advertise the availability information of cached content to its peering domains.

Fig. 9 illustrates the basic operations of ICC. Assume that A and B are two peering domains but C is the provider of A and B in terms of AS-level relationship. For simplicity, we assume that a domain has only one RH which has the capability of caching DATA packets. We also assume that the reachability information of the content hosted by the source has been propagated to the network in a way described in Section II-B. Accordingly, when a user (e.g., E) wants to obtain the content, he sends out a FIND packet that will be forwarded to the content source, as illustrated by (1) in Fig. 9. Then the content source responds the FIND packet and sends out the corresponding DATA packets to the network. The DATA packets will be forwarded to E along the reverse path of the FIND packet, as illustrated by (2) in Fig. 9. Each RH along the path caches a copy of the DATA packet and advertises the availability information to its peering domains, as illustrated by (3) and (4) in Fig. 9, respectively. As a result, RH_B knows that RH_A has a cached copy of the content. Later when RH_B receives a locally-unsatisfied FIND packet

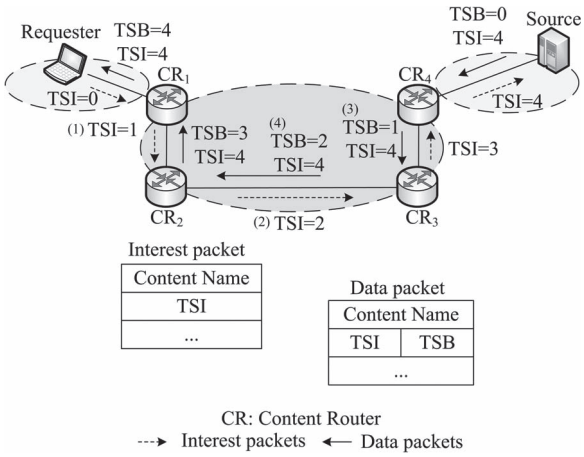


Fig. 10. Illustration of Probcache operations.

for the content, it forwards the FIND packet to RH_A instead of RH_C, as illustrated by (5) in Fig. 9. We can see that ICC can reduce the downloading traffic from its provider networks and improve the cache efficiency.

ICC is classified as homogeneous caching and on-path caching since the DATA packet is cached at every RH along the path from the node answering a FIND packet to the requester. ICC is a cooperative caching mechanism that makes the cached contents available for off-path requests, or more specifically, it is an inter-domain cooperative caching mechanism because the RHs in ICC are required to advertise the cache states to their peering domains. Since the RHs can decide where to cache a requested Data packet, both DONA and ICC are able to support the off-path caching which utilizes the cache resource in a more efficient way. Note that it is still lack of optimization methods for a RH to choose one or more content routers to cache DATA packets at different optimization conditions.

D. Heterogeneous and Cooperative Caching

1) Probcache: Probcache is another probabilistic caching mechanism. In particular, it adds a *Time Since Inception* (TSI) field in the Interest packet header, and adds a TSI field and a *Time Since Birth* (TSB) field in the Data packet header. When a user sends out an Interest packet, the TSI value is set to be zero. When a content router receives an Interest packet, it increases the TSI value of the Interest packet by one, as illustrated by (1) and (2) in Fig. 10, respectively. Similarly, when a content source sends out a Data packet, it sets the TSB value of the Data packet to be zero and sets the TSI value of the Data packet being the one carried in the corresponding Interest packet. In contrast, when a content router receives a Data packet, it increases the TSB value of the Data packet by one, while keeping the TSI value unchanged, as illustrated by (3) and (4) in Fig. 10, respectively.

In Probcache, the $TimesIn$ factor is employed to estimate the remaining caches that are still available on the downloading path. Specifically, when a given content router CR_x receives a Data packet, it calculates the $TimesIn$ factor by using Eq. (1),

$$Timesin(x) = \frac{\sum_{i=1}^{c-(x-1)} N_i}{T_{tw} N_x} \tag{1}$$

where c and x represent the TSI value and TSB value in the header of a Data packet, respectively. The value of T_{tw} is defined as the target time window and is set to be 10 in [32]. The values of N_i and N_x represent the average caching capacity along the forwarding path and the CR_x's caching capacity, respectively. The sum in Eq. (1) is used to account for the remaining caching capacity of the downstream content routers along the path. The $TimesIn$ factor implies the number of times that the Data packet may be cached by the downstream content routers along the path.

When the Data packet arrives at the content router, the TSI value and the TSB value are used to calculate the $CacheWeight$ factor by using Eq. (2).

$$CacheWeight(x) = \frac{TSB(x)}{TSI(x)} \tag{2}$$

For example, when CR_4 in Fig. 10 receives the Data packet, it increases the TSB value by one. Thus, the TSI value is four and the TSB value is one. Accordingly, the $CacheWeight$ factor is 0.25 ($= 1/4$). Similarly, when CR_3 receives the Data packet, it increases the TSB value by one, and the resulting TSB value is two. As a result, the $CacheWeight$ factor is 0.5 ($= 2/4$). From Eq. (2), it is clear that the closer a Data packet approaches the destination, the higher is the value of the $CacheWeight$ factor.

Having computed the $TimesIn$ factor (i.e., $TimesIn(x)$) and the $CacheWeight$ factor (i.e., $CacheWeight(x)$), a content router caches an incoming Data packet with probability $Probe(x)$, which is calculated by using Eq. (3).

$$Probe(x) = Timesin(x) \times CacheWeight(x) \tag{3}$$

Similar to Prob(p), Probcache is also aimed at reducing the caching redundancy and tends to cache the popular contents. In addition, both of them are heterogeneous and on-path caching mechanisms. However, the content routers in Probcache make the caching decision through a cooperative mechanism because the TSI value and the TSB value are provided by content routers along the path. Thus Probcache is a cooperative caching mechanism and it uses an implicit cooperation because it dose not require the content routers to advertise their cache states. It is also important to note that the cached contents in Probcache can be used only for on-path requests since the content routers can forward requests only according to the FIB.

2) CBC: CBC is aimed at reducing the caching redundancy. It expands the work of CEE and caches the Data packet at the "important" content routers that have the highest betweenness centrality value along the downloading path. The betweenness centrality of a router reflects how often the router lies in the shortest paths between any two of the other routers. Consider a bidirectional network shown in Fig. 11, where we assume that the network is administrated by a centralized topology manager. The betweenness centrality value of each content router (denoted by $BC(R_y)$) is pre-calculated offline by using Eq. (4):

$$BC(R_y) = \sum_{\forall R_i, R_j \in V \setminus R_x} \frac{n(R_i, R_j, R_y)}{n(R_i, R_j)} \tag{4}$$

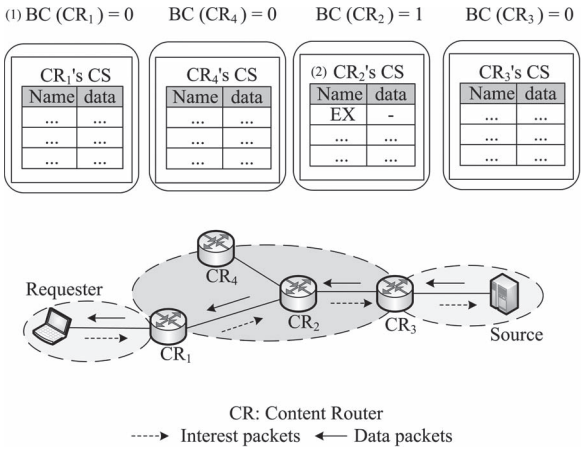


Fig. 11. Illustration of CBC operations. BC stands for Betweenness Centrality.

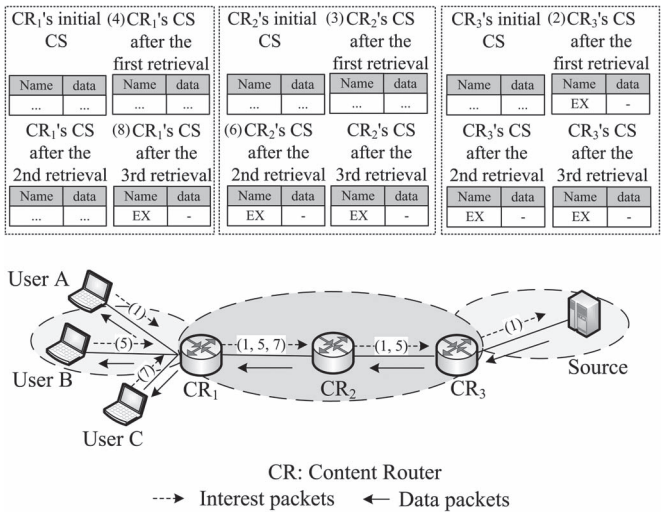


Fig. 12. Illustration of LCD operations.

where $n(R_i, R_j)$ is the number of shortest paths between R_i and R_j while $n(R_i, R_j, R_y)$ represents the number of shortest paths from R_i to R_j through R_y.

For example, the betweenness centrality values of the content routers in Fig. 11 are shown by (1) in the figure. When the requester issues an Interest packet, CBC adds a field in the Interest packet header to record the maximum betweenness centrality value. Accordingly, when a content router forwards an Interest packet, it compares its betweenness centrality value with the one carried in the Interest packet. If its betweenness centrality value is larger than the one carried in the Interest packet, the content router replaces the later by its betweenness centrality. When the content source receives the Interest packet, it copies the betweenness centrality carried in the Interest packet and records the betweenness centrality value into the corresponding Data packet. Accordingly, when a content router receives a Data packet, it compares its own betweenness centrality value with the one carried in the Data packet. If they are equal, the content router caches the Data packet into its CS. This way, the Data packet would be cached at the content router whose betweenness centrality is the largest among the content routers along the path. In the example shown in Fig. 11, the content router that has the largest betweenness centrality along the path $CR_1 \rightarrow CR_2 \rightarrow CR_3$ is CR_2 since its betweenness centrality is one. Accordingly, CR_2 will cache the Data packet sent from the source to the requester, as illustrated by (2) in Fig. 11. Note that if there are multiple content routers whose betweenness centrality matches with the one carried in the Data packet, the Data packet is cached at all such content routers. While CBC can reduce the caching redundancy, it does not consider the present cache state of the network, which makes Data packets to be cached at the important content routers along the path, thus causing frequent replacements at the important content routers and leaving other content routers idle.

From the above descriptions, one can see that CBC is a cooperative caching mechanism because the content routers decide whether to cache the arrived Data packets by means of implicit cooperation and the cached contents are used only for on-path requests. It is also a heterogeneous caching mechanism since content routers are not required to cache every arrived Data packet. Furthermore CBC falls into the on-path caching category since the Data packet can only be cached along its delivery path.

3) LCD and MCD: LCD is a new method to realize the heterogeneous caching through a cooperative way and aimed at reducing the caching redundancy. In LCD, a caching suggestion flag bit is added in the Data packet header. When a request arrives at the content source or at the cache that holds a copy of the corresponding content, the flag bit is activated and then the content routers along the downloading path use this bit to determine whether they should cache the Data packet or not. This process is illustrated by using the example shown in Fig. 12, where we assume that a user (e.g., A) wants to obtain a content hosted by the source. For this purpose, A sends out an Interest packet to the network (arrow 1). When the content source receives the Interest packet, it responds with a Data packet whose caching suggestion flag bit is set to one. When the first downstream content router CR_3 receives the Data packet and finds that the flag bit is activated, it caches the Data packet in its CS, as illustrated by (2) in Fig. 12. After that, CR_3 resets the caching suggestion flag bit to zero and sends the Data packet to the next-hop content router CR_2. When CR_2 receives the Data packet, it finds that the caching suggestion flag bit is zero and does not cache the Data packet, as illustrated by (3) in Fig. 12. Similarly, when CR_1 receives the Data packet and finds that the caching suggestion flag bit is zero, it forwards the Data packet to A without caching the Data packet, as illustrated by (4) in Fig. 12. When another user (e.g., B) wants to obtain the same content, it sends out an Interest packet to the network (arrow 5). Now CR_3 satisfies the Interest packet and sets the caching suggestion flag bit to be one in the corresponding Data packet. When CR_2 receives the Data packet and finds that the caching suggestion flag bit is one, it caches the Data packet and sets the flag bit to be zero, as illustrated by (6) in Fig. 12. Accordingly, CR_1 will not cache the Data packet. Note that only after the third retrieval issued by C (arrow 7), the content will be cached at CR_1, as illustrated by (8) in Fig. 12. From the above descriptions, we can see that LCD can cache popular contents at

the network edge while preventing the unpopular content from occupying cache spaces. However, the popular content would be cached at the network edge only after several retrievals. In addition, a content may be cached at many content routers, thus cannot efficiently address the caching redundancy problem.

MCD is proposed to address the problem of caching redundancy in LCD. MCD works in a way much similar to LCD. The difference is that when a content router caching a copy of a desired content receives an Interest packet and forwards the corresponding Data packet to its downstream content router, it also deletes the cached copy. This way, the content will be cached only at one content router along the path between the content source and the requester, thus efficiently reducing the caching redundancy.

Both LCD and MCD are classified into the cooperative caching category because the content routers make caching decisions by means of implicit cooperation, so that the cached contents in LCD and MCD can only be used for on-path requests. In addition, they belong to the heterogeneous caching since the content routers are not required to cache each arrived Data packet. Furthermore, they are on-path caching because the Data packet can only be cached at the content routers along the delivery path.

4) WAVE: As a heterogeneous caching, WAVE adds a caching suggestion flag bit in Data packets to help content routers make caching decisions. In ICN, a large content is divided into many small chunks. For example, a content EX can be divided into five chunks, named by EX/C_1, EX/C_2, EX/C_3, EX/C_4, and EX/C_5, respectively. To obtain EX, a requester issues five Interest packets, each of which corresponds to one chunk of the content.

WAVE works much similar to LCD. The most significant difference is that WAVE treats the chunks of a same content in a cohesive manner, but LCD deals with each chunk individually. In particular, while the node answering an Interest packet for a content sets the caching suggestion flag bit of every content chunk to be one in LCD, it only sets the caching suggestion flag bits of some content chunks to be one in WAVE. The number of content chunks to be set can be calculated through the chunk caching algorithm, as illustrated in Algorithm 1.

Algorithm 1 Chunk Caching Algorithm. CMW stands for Chunk Marking Window.

Input:
1: b: CMW base (e.g., 1, 2, 3...);
2: s: CMW state (initial value $= 0$);
3: c: index of the last cached chunk at R_i;
4: l: index of the last cached chunk at R_i's downstream content router (initial value $= 0$);
5: r: index of the requested chunk;
Output:
6: marked chunks
7: **if** $r \leqslant l$ **then**
8: mark the requested chunk r
9: $l \leftarrow r$
10: $s \leftarrow \lfloor \log_b r \rfloor$
11: **else if** $l < r \leqslant \sum_{j=0}^{s} b^j$ **then**
12: mark the requested chunk r
13: $l \leftarrow r$
14: **end if**
15:
16: forward the requested chunk r to the next hop
17:
18: **if** $r == c$ **then**
19: s++
20: **end if**

The *Chunk Marking Window Base* (CMW base, which is denoted by b) determines the speed of chunk caching and it is set in advance by the network administrator. In the chunk caching algorithm of WAVE, both the CMW base and a content router's current *Chunk Marking Window State* (CMW state) determine the number of content chunks to be marked by this router. We assume that the CMW base b is set to be two and there are k retrieval requests for the same content. The content routers along the path from the content source to the requester are sequentially expressed as $R_0, R_1, R_2, \ldots$, and R_m, where m represents the number of nodes (including the source) along the path. When the content source receives the first content retrieval request, its initial value of the CMW state (denoted by s) and the index of the latest cached chunk at the downstream (denoted by l) are both zero. The content source sets the caching suggestion flag bit of the first chunk to be one (line 12) and the value of l to be the index of the requested chunks (line 13). In addition, it sets the flag bits of the rest chunks to be zero and increases the value of s by one when the last chunk of the content has been delivered (line 19). If the downstream content router R_1 receives a chunk whose caching suggestion flag bit is equal to one, it caches the chunk, resets the caching suggestion flag bit to be zero, and forwards the chunk toward the requester. Moreover, R_1 sets the initial value of s to be zero. As a result, only the 1st content chunk is cached by R_1.

For the second content retrieval request, the content source sets the caching suggestion flag bits of the $(l+1, \sum_{j=0}^{s} 2^j)$ chunks to be one (i.e., the 2nd and 3rd chunks, because both the current values of l and s are one), but sets those of the other chunks to be zero. R_1 answers the Interest packet with the 1st content chunk, and sends out the 1st content chunk after setting its caching suggestion bit to be one. In addition, it sets the value of l to be one, and increases the value of s by one since the last cached chunk of the content has been forwarded to the next hop. If R_1 receives the marked chunks (i.e., the 2nd and 3rd chunks), it resets the flag bit to be zero and forwards them after caching them. In general, for the i-th $(1 < i \leqslant k)$ content retrieval request, the content source sets the caching suggestion flag bits of the $(l+1, \sum_{j=0}^{i-1} 2^j)$ chunks to be one, but sets those of the other chunks to be zero.

We illustrate the above process with the help of Fig. 13, where we assume that the content source holds a content named EX which comprises five chunks, EX/C_1, EX/C_2, EX/C_3, EX/C_4, and EX/C_5. We also assume that the requester (e.g., A) wants to obtain EX. For this purpose, the requester sends out five Interest packets to the content source. When the source receives the Interest packets, it sets the caching suggestion flat bit of the first content chunk (i.e., EX/C_1) to be one, but sets

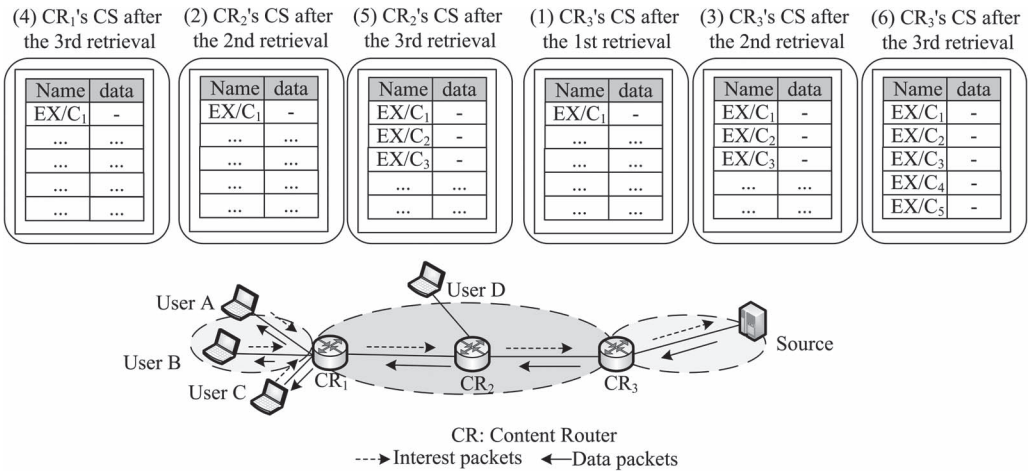


Fig. 13. Illustration of WAVE operations.

the caching suggestion flat bits of the rest four content chunks to be zero. As a result, CR_3 caches the first content chunk since it finds that the caching suggestion flat bit of the first content chunk is one, as illustrated by (1) in Fig. 13. After that, CR_3 resets the caching suggestion flat bit to be zero and forwards it to the next-hop content router CR_2. On the other hand, when CR_3 receives the rest four content chunks, it simply forwards them to CR_2 without caching them since their caching suggestion flag bits are zero.

When another user B sends out the Interest packets to obtain the same content, CR_3 will send the first chunk to the requester. On the other hand, the other four chunks will be forwarded by the content source to the requester. Note that, when CR_3 sends out the first chunk, it will set the caching suggestion flag bit to be one. Thus, CR_2 will cache the first chunk, as illustrated by (2) in Fig. 13, and resets its caching suggestion flag bit to be zero. Accordingly, CR_1 will not cache the first chunk. Different from the Interest packet for the first chunk, the Interest packets for other chunks will be responded by the source node. Now, the source node sets the caching suggestion flag bit of the 2nd and the 3rd chunks to be one, while setting the caching suggestion flag bit of the rest chunks to be zero. Therefore, the 2nd and 3rd chunks will be cached at CR_3, as illustrated by (3) in Fig. 13. Note that the rest content chunks will not be cached along the path.

When a given user C sends out the Interest packets to obtain the same content, CR_2 will answer the Interest with the first chunk and sets the caching suggestion flag bit of the corresponding Data packet to be one. Thus CR_1 will cache the first chunk, as illustrated by (4) in Fig. 13. Similarly, CR_3 will answer the Interests with the 2nd and 3rd chunks and sets the caching suggestion flag bit of the corresponding Data packets to be one. Accordingly, CR_2 will cache the 2nd and 3rd content chunks into its CS, as illustrated by (5) in Fig. 13. On the other hand, the source will answer the Interests with the 4th and 5th chunks and sets the caching suggestion flag bit of the corresponding Data packets to be one. As a result, CR_3 will cache the 4th and 5th content chunks in its CS. The resulting CS of CR_3 is illustrated by (6) in Fig. 13.

We can observe that CR_2's l and s are both one after the third retrieval. Note that, if CR_2 receives the Interest packets for the same content chunk from D, it will answer the Interests with the 1st, 2nd and 3rd chunks, but only set the caching suggestion bit of first chunk to be one. Particularly, when CR_2 receives the Interest packets for the first content chunk, it will set the caching suggestion bit of the first chunk to be one and return the chunk to requester. Then it resets the values of l to be one (line 9) and s to be zero (line 10), respectively. When CR_2 receives the Interest packets for the 2nd and 3rd content chunks, it sets the flag bit of the corresponding chunks to be zero and forwards them to the requester.

WAVE and LCD have the same advantages and disadvantages. For example, they can improve the cache efficiency by reducing caching redundancy. In addition, they can cache more content under the limited cache resources and enhance the cache utilization efficiency. However, popular contents can be cached at content routers close to users only after several rounds of content retrievals. Similarly, the shared information about cache states is narrowed to help content routers make caching decisions at the arrival of the Data packet. Once the Interest packet arrives, caches are converted into independent work mode.

From the above descriptions, one can see that WAVE is a cooperative caching mechanism. More specifically, it uses an implicit cooperation since the content routers in WAVE do not need to advertise their cache states to the other content routers. In addition, WAVE is both a heterogeneous and on-path caching in which the cached contents are used only for on-path requests.

5) CLS: CLS is an improved version of MCD, aiming at allowing off-path requests. In particular, a content router in CLS maintains a caching table that comprises many entries, each of them is a 4-tuple $(Name, NodeID_{in}, NodeID_{out}, Hop)$ representing the content name, the upstream content router, the downstream content router and the distance to the content source, respectively.

We now illustrate the basic operations of CLS with the help of Fig. 14. When the first downstream content router on the path between the source and the requester receives the Data packet, it will hold a copy and generate a caching trail $(EX, null, null, 1)$, in which the first and the second $nulls$ indicate that the upstream node is the source and the content is cached locally, respectively, as illustrated by (1) in Fig. 14. The other

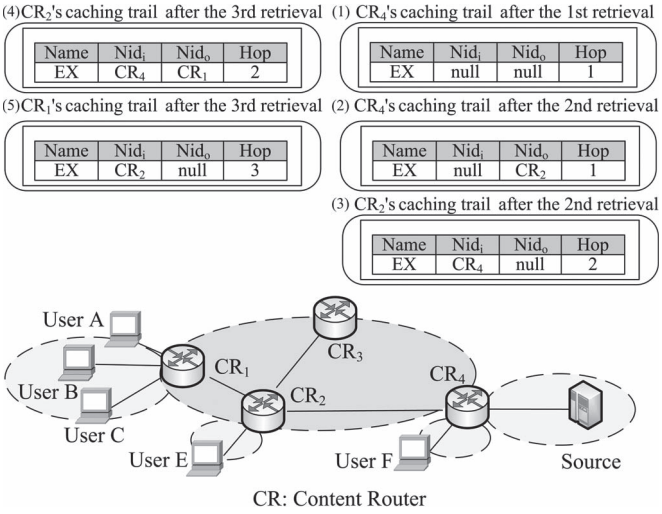


Fig. 14. Illustration for the basic operation of CLS. Nid_i stands for $NodeID_{in}$. Nid_o stands for $NodeID_{out}$.

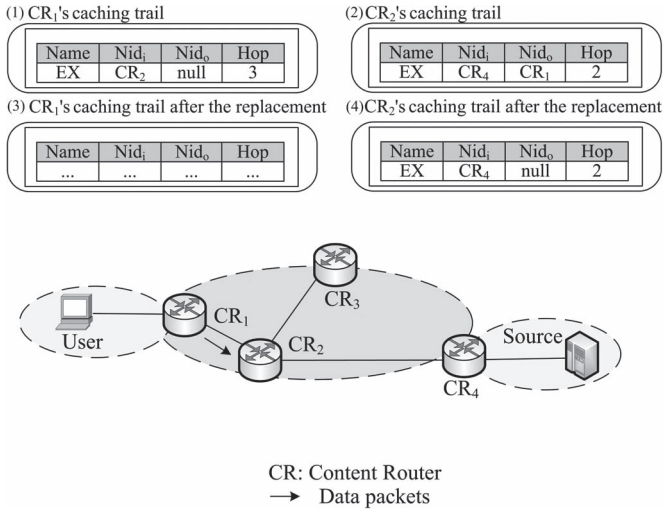


Fig. 15. CLS replacement process. Nid_i stands for $NodeID_{in}$. Nid_o stands for $NodeID_{out}$.

content routers on the downloading path do not cache the content to avoid redundant replications.

When the second request for the same content reaches CR_4, CR_4 satisfies the Interest packet and then evicts the content from its CS. Meanwhile, the caching trail is modified to $(EX, null, CR_2, 1)$, which indicates that the downstream content router on the forwarding path is CR_2, as illustrated by (2) in Fig. 14. CR_2 caches the Data packet in its CS and generates a caching trail $(EX, CR_4, null, 2)$, as illustrated by (3) in Fig. 14, which means that the upstream content router is CR_4 and the content router is two-hop from the source. On the other hand, CR_1 does not cache the content but simply forwards it to the requester.

Since CR_2 locally caches the Data packet, when CR_2 receives the third Interest packet for the same Data packet, it satisfies the Interest packet by sending the corresponding Data packet to CR_1. In addition, CR_2 modifies the caching trail for the content name to be the one shown in (4) in Fig. 14. Note that the third field changes from $null$ to CR_1. When CR_1 receives the Data packet, it locally caches the Data packet and records a caching trail for the corresponding content name, as illustrated by (5) in Fig. 14.

The caching trail in CLS indicates the caching location of the corresponding content. According to the above description, the value of the Hop field in a caching trail indicates the number of hops from the content source to the present content router. If no copy of the content exists in a content router's CS but this router can find a caching trail when an Interest packet arrives, the Hop value in the trail for the name can be used to decide the next hop of the Interest packet, possibly the upstream content router or the downstream content router. When the value is not less than a threshold value which has been set in advance by the network administrator (denoted by Th_b), the Interest packet is forwarded to the downstream content router recorded in the trail. Otherwise, it is forwarded to the upstream content router.

For example, we assume that the threshold value Th_b is predefined to be two. Thus, CR_2 forwards the locally-unsatisfied Interest packet for EX issued by E to its downstream content router (i.e., CR_1), since the value of the Hop field in the corresponding trail is not less than Th_b. While CR_4 forwards the locally-unsatisfied Interest packet for EX from F to its upstream content router (i.e., the content source) because the value of the Hop field is one.

When the cached copy is evicted by a content router, this router sends the copy to the upstream content router recorded in the caching trail for the corresponding content name. Assume that the caching trails at CR_1 and CR_2 are shown by (1) and (2) in Fig. 15. When CR_1 evicts the cached content EX and deletes the corresponding caching trail, the content will be forwarded to the upstream content router CR_2 and cached at CR_2, as illustrated by (3) in Fig. 15. When CR_2 receives the content, it changes the value of the $NodeID_{out}$ field from CR_1 to $null$, as illustrated by (4) in Fig. 15. On the other hand, the caching trail for the content at CR_4 keeps unchanged.

In this way, a content router can decide the destination (possibly a cache) that the Interest packet should be forwarded to when it does not have a copy of the content. Different form MCD in which the cached content is available only for on-path requests, CLS supports the off-path availability by establishing caching trails. As a result, CLS further improves the cache resource utilization while remaining the advantages of MCD, but at the cost of additional expenses for maintaining and updating the caching trails.

Similar to MCD, CLS uses an implicit cooperation since content routers are not required to advertise their cache states. In addition, it falls into the heterogeneous caching category because the content routers do not cache every arrived Data packet. It is also an on-path caching mechanism since the Data packet can only be cached at the content routers along the delivery path.

6) CINC: CINC is a cooperative caching mechanism designed for large video streams (e.g., live TV services). For a network with q content routers, CINC assigns a unique label (ranging from 0 to $q-1$) to every content router in the network. In addition, CINC marks Data packets of a video with sequence numbers. Let L_r be the label of a content router and S_d be the sequence number of a Data packet. Furthermore, if we divide

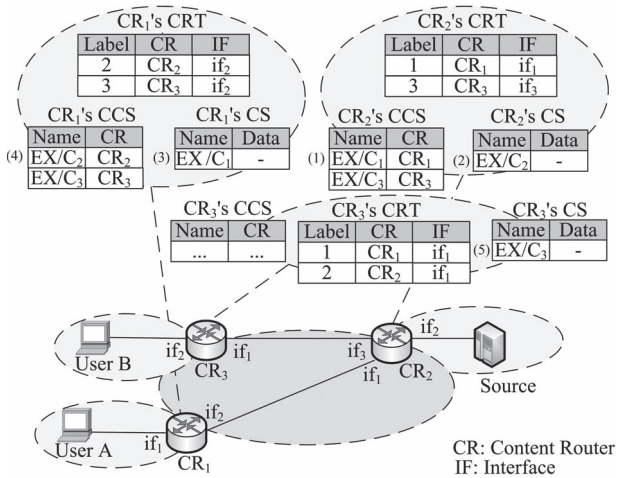


Fig. 16. Illustration of CINC operations.

S_d by q, we denote z be the modulus of the division. To avoid caching redundancy, a Data packet with sequence number S_d is only cached at the content router whose label equals to z. To achieve this, CINC adds two new tables in each content router: a *Collaborative Router Table* (CRT) that records the reachability information of other content routers in the network, and a *Collaborative Content Store* (CCS) that records the names of the contents cached by other content routers. The CRT of a content router maintains an entry for every content router (except the router itself) in the network. The entry records the label assigned to the content router, the identifier of the content router, and the incoming interface that the reachability information for the content router arrives at. When a content router receives a Data packet with sequence number S_d, the content router stores the Data packet into its local CS if its label equals to z. Otherwise, it forwards the Data packet to the content router whose label equals to z, and adds an entry for the Data packet into its CCS table. When the content router whose label equals to z receives the Data packet, it stores the Data packet into its local CS.

Fig. 16 illustrates the above process, assuming that a given user A wants to obtain the content named by EX which is published by the content source and is comprised of three chunks (i.e., EX/C_1, EX/C_2, and EX/C_3). In addition, we assume that content routers CR_1, CR_2 and CR_3 are assigned labels 1, 2, and 3, respectively. The CCSs of all content routers are initially null and the corresponding CRTs of CR_1, CR_2, and CR_3 are illustrated in Fig. 16. When CR_2 receives the first chunk (i.e., EX/C_1), it finds that it should not locally cache the content chunk because its label (i.e., $L_r = 2$) does not match with the sequence number of the content chunk (i.e., $S_d = 1$). Accordingly, it forwards the chunk to CR_1 based on its CRT and finds a match in its PIT. Since the next hop for the packet is also CR_1, CR_2 forwards the chunk to CR_1 and deletes the PIT entry for content chunk. Moreover, CR_2 adds an entry for the first content chunk into its CCS, so that the subsequent Interest packets requiring the same content chunk will be forwarded to CR_1, as illustrated by (1) in Fig. 16. When the second content chunk (i.e., EX/C_2) reaches CR_2, CR_2 caches it locally, forwards it to CR_1 according to the PIT, and deletes the corresponding entry in the PIT, as illustrated by (2) in Fig. 16. In addition, when the third content chunk (i.e., EX/C_3) arrives at CR_2, CR_2 finds that its label does not match with the sequence number of the third content chunk. Accordingly, it looks up its CRT, and then forwards the third content chunk to CR_3 because CR_3's label matches with the sequence number of the third content chunk. After forwarding the third content chunk to CR_1 based on its PIT, CR_2 deletes the entry for the third content chunk from its PIT and adds an entry for the third content chunk into its CCS. In summary, CR_2 caches the matching content chunk (i.e., EX/C_2), sends the content chunks (i.e., EX/C_1 and EX/C_3) to the other content routers based on its CRT, and forwards the three content chunks to CR_1 according to its PIT. As a result, CR_2 adds entries for the first and the third content chunks into its CCS.

When CR_1 receives the first content chunk, it caches the chunk and forwards the chunk to A based on its PIT, as illustrated by (3) in Fig. 16. When CR_1 receives the second content chunk or the third content chunk (i.e., EX/C_2 or EX/C_3), it finds that its label does not match with the sequence number of the content chunk. As a result, it sends the replicated chunk to the matching content router (i.e., CR_2 or CR_3) according to its CRT. Then CR_1 forwards the content chunks to the user based on its PIT. Finally, it adds an entry for EX/C_2 and one for EX/C_3 into its CCS, as illustrated by (4) in Fig. 16. Since CR_2 has received the content chunks, it will cause redundancy. In CINC, nonce is required to prevent broadcast storm. In details, when the duplicated packet (i.e., EX/C_2) with the same nonce is received by CR_2, it should be immediately discarded.

When CR_3 receives the third content chunk, it locally caches the third content chunk, as illustrated by (5) in Fig. 16. From the above descriptions, it is clear that CINC supports off-path caching because CR_3 is not on the path from the content source to the user but caches the third content chunk.

When an Interest packet arrives, a content router deals with the Interest packet by sequentially checking its CS, CCS, PIT and FIB. If one matches, the process terminates. For example, B sends out an Interest packet for EX/C_1, an Interest packet for EX/C_2, and an Interest packet for EX/C_3. When CR_3 receives the Interest packet for EX/C_1, it cannot find a cached copy for EX/C_1 in its CS. Accordingly, it queries its CCS but cannot find an entry for EX/C_1. As a result, CR_3 forwards the Interest packet to CR_2 after querying its FIB. Similarly, CR_3 forwards the Interest packet for the second content chunk to CR_2. On the other hand, since CR_3 has cached the third content chunk forwarded from CR_1, when CR_3 receives the Interest packet for the third content chunk, it directly forwards the third content chunk to B. Accordingly, CR_2 answers the Interest packet for EX/C_2 with the corresponding content chunk and forwards the Interest packet for EX/C_1 to CR_1 based on its CCS.

It is possible that a content chunk may be evicted at the content router whose label matches with the sequence number of the content chunk. To address this issue, CINC adds a flag bit in the Interest packet header. When the Interest packet arrives at a content router in a network for the first time, the content router sets the flag bit to be zero and forwards the packet according to the CCS if there is an entry for the name in the CCS table. If the content router whose label matches with the sequence number

of the corresponding content chunk caches the desired content chunk, it answers the Interest packet with the corresponding content chunk. Otherwise, it sets the flag bit to be one and sends the Interest packet to the next hop according to its FIB. When subsequent content routers receive the Interest packet, they find that the flag bit is set to be one and will forward the Interest packet based on their FIB, without querying their CCS tables. Moreover, they delete the corresponding entry in their CCS tables if they can find one for the content name.

CINC is an off-path caching mechanism because the Data packet can be forwarded to and then cached at the content routers that are not along the delivery path. In addition, it is a heterogeneous caching mechanism since the content routers do not cache every received Data packet. Meanwhile, CINC uses an implicit cooperation and the additional CCS table makes the cached content available for off-path requests.

7) CPHR: CPHR is aimed at reducing caching redundancy through cooperative caching. For this purpose, a content router in CPHR adds a new field called *Egress Router* to every entry of its FIB to record the egress router that an Interest packet for the content name leaves the domain toward the content source. In addition, an ingress router puts two additional fields in the Interest packet header. A *Cache Name* field is used to record the caching location of the content chunk corresponding to the content name in the Interest packet, and an *Egress Router* field indicates the name of the egress router for the content name in the Interest packet. CPHR also uses consistent hashing to map content names to content routers in the network. In particular, assuming that a content name is hashed to a content router (denoted by R_z), the content chunk corresponding to the content name is cached at R_z. Similarly, when a content router receives an Interest packet, it hashes the content name carried in the Interest packet and forwards the Interest packet to the content router that the content name hashes to, with the help of the *Cache Name* field carried in the Interest packet. If that content router locally caches the corresponding content chunk, it satisfies the request; Otherwise, it forwards the Interest packets toward the content source, with the help of the *Egress Router* field carried in the packet header.

We illustrate the basic operations of CPHR with the help of Fig. 17, assuming that the ingress router IR_1's FIB is shown by (1) in Fig. 17 and a content (e.g., EX) is provided by the content source. When IR_1 receives the Interest packet, it hashes the content name to a content router (e.g., CR_3). Accordingly, it queries its PIT but cannot find an entry for the content name. As a result, IR_1 stores an entry for the content name in its PIT, sets the *Cache Name* field of the Interest packet to be CR_3 and the *Egress Router* field to be ER_3 based on its FIB. In addition, IR_1 prepends CR_3 onto the content name EX, thus forming a new name CR_3/EX. After that, IR_1 forwards the Interest packet toward CR_3, as illustrated by (2) in Fig. 17. Moreover, CR_2 does not need to prepend CR_3 onto the content name because it finds that the cache name matches with CR_3, and then it forwards the Interest packet toward CR_3 by using the new name CR_3/EX. Note that the name picked up by CR_2 is the new one (i.e., CR_3/EX), while the original name (i.e., EX) should be recorded in the PIT entry. Therefore it is necessary that CR_2 delete the prefix with the help of the

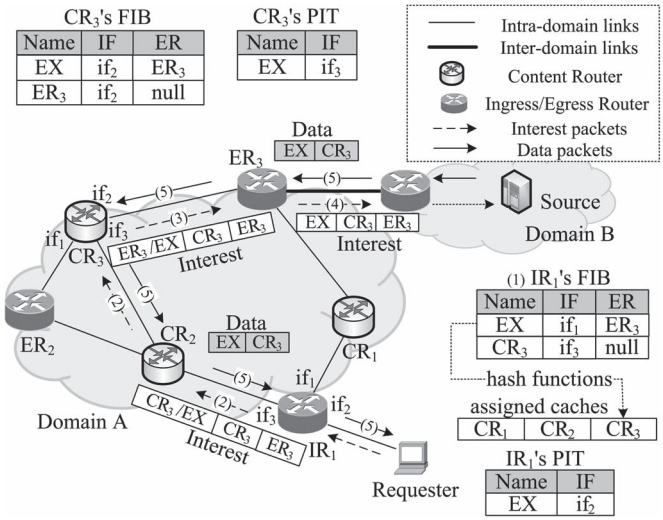


Fig. 17. Illustration of CPHR operations. IF stands for Interface. ER stands for Egress Router. IR stands for Ingress Router.

Cache Name field. When CR_3 receives the Interest packet, it satisfies the Interest packet if it can find a cached copy of the desired content. Otherwise, it also deletes the *Cache Name* part from the name carried in the Interest packet and adds an entry for EX into its PIT. Then CR_2 forwards the Interest packet toward the content source. For this purpose, it prepends the *Egress Router* field onto the content name, thus forming a new name ER_3/EX. After that, it forwards the Interest packet to ER_3, as illustrated by (3) in Fig. 17. When ER_3 receives the Interest packet, it finds that it is the egress router of the Interest packet. Accordingly, it deletes the *Egress Router* part from the name carried in the Interest packet, records the PIT entry, and forwards the Interest packet toward the content source, as illustrated by (4) in Fig. 17.

When the egress router receives the corresponding Data packet, it directly forwards the Data packet to the next hop based on its PIT. When CR_3 receives the Data packet, it finds that the content name hashes to its own label. Accordingly, it caches the Data packet and forwards the Data packet toward the user based on its PIT, as illustrated by (5) in Fig. 17.

From the above descriptions, it is clear that CPHR is an on-path caching mechanism since a Data packet is cached along its delivery path. In addition, CPHR is a cooperative caching mechanism in which the content routers cache the content chunks in a cooperative manner. It is also a heterogeneous caching mechanism since the content routers are not required to cache every incoming Data packet. The content routers forward the Data packet in terms of the new content name after hashing, which makes the cached content in CPHR available for off-path requests.

8) LCC: LCC is a hash-based caching mechanism for inter-domain cooperation. Similar to CPHR, LCC hashes the content name to a unique content router within a domain, and uses the hash function to map the content to the chosen content router. Then, this router becomes the only one that caches this particular content. In LCC, a redundancy control scheme is proposed. To achieve this, a domain negotiates with the neighbors about which content it would like to cache. For example, we mark the set of contents that a domain wants to cache as $[a, b]$ and

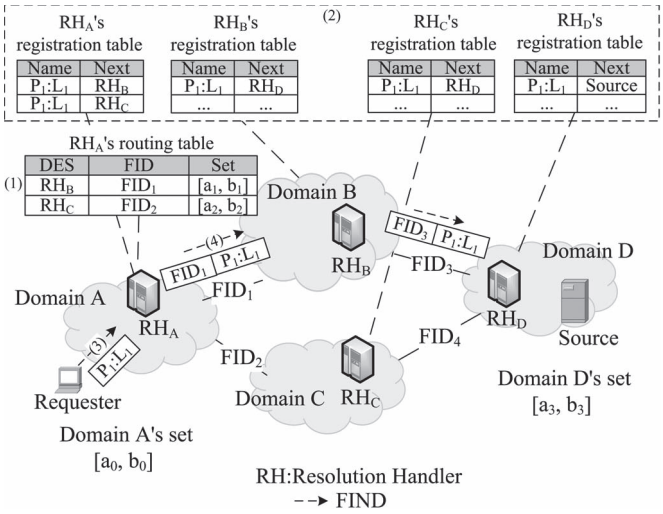


Fig. 18. Illustration for content retrieval in LCC. DES stands for Destination.

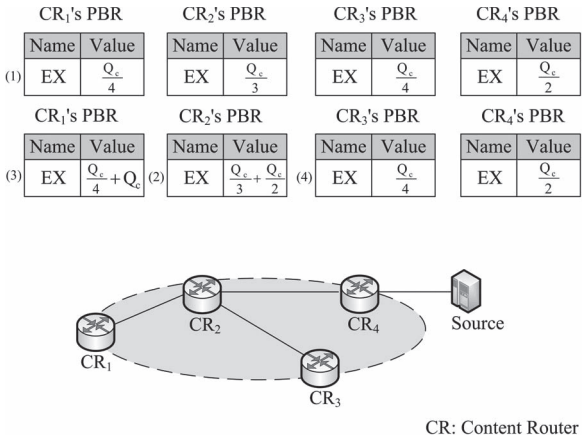


Fig. 19. Illustration of CATT operations. PBR stands for Potential Based Routing.

use the $hash(name)$ to indicate the hash value of the content name. If $a \leqslant hash(name) \leqslant b$, the content will be cached by the domain.

We use Fig. 18 to illustrate the content retrieval and caching in LCC based on DONA, although it is not limited to such an ICN architecture. LCC adopts the Pathlet routing [44], which is a FID-based (Forwarding Identifier) multipath routing scheme, to select the inter-domain path. In Fig. 18, we assume that there are four domains (i.e., A, B, C and D) which have connected to each other and learned the corresponding FIDs between them. Each of them adds a pathlet entry in an additional routing table for recording the destination and the corresponding FID. As illustrated by (1) in Fig. 18, the entry in the routing table instructs A to forward packets to B through the inter-domain path FID_1. After one-hop pathlets are constructed, multihop pathlets can be built by using them. For example, A sets up a pathlet toward D ($FID_1 \rightarrow FID_3$). Since we are more concerned with the in-network caching mechanisms, more details about Pathlet routing can be obtained from the paper [44].

LCC recommends that each domain should advertise the boundaries of potential cached content to its neighbors when its own FIDs are updated. For example, when B learns the FIDs of its neighbors, it will retrieve and propagate updates on the contents that the neighbors would like to cache, and add what it learns into the routing table. For example, the boundaries of content cached by A and B are continuous sets $[a_1, b_1]$ and $[a_2, b_2]$, respectively. Note that, two neighboring domains should not cache the same contents. As illustrated by (2) in Fig. 18, we assume that the RHs have received the REGISTER message of $P_1 : L_1$ from the content source. When RH_A receives a FIND for $P_1 : L_1$, it discovers two pathlets $FID_1 \rightarrow FID_2$ and $FID_3 \rightarrow FID_4$. Due to a lack of solutions about how to negotiate the boundaries of potential cached content in the literature [37], we assume that $100 \leqslant hash(P_1 : L_1) \leqslant 200$. Then RH_A places FID_1 in the packet header and forwards the packet to RH_B through the path FID_1, as illustrated by (3) and (4) in Fig. 18. When RH_B receives the FIND packet, it hashes the content name, and forwards the packet to the matching content router after finding that the corresponding DATA packet is its interest to cache. If a cache hit occurs then the DATA packet goes back to the requester, otherwise the FIND packet should be forwarded to the next domain. The corresponding DATA packet is delivered along the reverse path of the FIND packet. Whenever a content router receives a DATA packet with a matching name it should hold a copy.

According to the above description, we conclude that LCC adopts additional advertisement mechanism and therefore belongs to explicit cooperation. Moreover, the Data packets are only cached on their delivery path, so LCC falls into on-path caching category. Similar to CPHR, LCC makes the cached content available for off-path requests.

9) CATT: CATT caches a Data packet at a single node along the downloading path in every network. In addition, CATT introduces the content's expected quality to help content routers forward Interest packets. If a content is cached or stored at a content router, the content's expected quality at this router is assumed to be Q_c. On the other hand, if a content is not cached at a content router, the content's expected quality at this router (denoted by $Qua(R_i)$) is calculated by using the following equation:

$$Qua(R_i) = \frac{Q_c}{Hop_r + 1} \tag{5}$$

where Hop_r is the number of hops from this router to the content router that caches or stores the content.

Fig. 19 illustrates the basic operations of CATT. In Fig. 19, the content source holds the desired content and advertises the reachability information together with the content's expected quality to other content routers in the network. Since CR_3 is one hop away from the content source, the content's expected quality at CR_3 is $\frac{Q_c}{2}$. Similarly, the content's expected quality at CR_1, CR_2, and CR_4 are $\frac{Q_c}{4}$, $\frac{Q_c}{3}$, and $\frac{Q_c}{4}$, respectively, because they are 3, 2, and 3 hops away from the content source, as illustrated by (1) in Fig. 19.

When a content router caches a content, it advertises the availability information of the content to its k-hop neighbors, where k could be one, two, three, or other numbers. When the neighbors receive the availability information of the content,

they firstly calculate the content's expected quality brought by the cached copy. By summing up the initial expected quality with the content's expected quality brought by the cached copy, each content router then obtains the content's new expected quality. In the example shown in Fig. 19, we assume that the content is cached at CR_1, which advertises the availability information of the content to its one-hop neighbor CR_2. Accordingly, the content's expected quality caused by the cached copy at CR_1 is $\frac{Q_c}{2}$. As a result, the content's new expected quality at CR_2 is $\frac{Q_c}{3} + \frac{Q_c}{2}$, as illustrated by (2) in Fig. 19. Note that since the content is cached at CR_1, its new expected quality at CR_1 should be $\frac{Q_c}{4} + \mathrm{Q}_c$, as illustrated by (3) in Fig. 19. On the other hand, its expected quality at CR_3 and CR_4 do not change because CR_1 only advertises the content's availability to its one hop neighbors, as illustrated by (4) in Fig. 19.

CATT uses a Potential Based Routing (PBR), instead of FIB, to retrieve the content. When a content router receives an Interest packet for the content but cannot find a cached copy in its CS, it compares the content's expected qualities of its neighbors' PBR and chooses the neighbor at which the content's expected quality is the largest to forward the Interest packet. For example, when CR_2 in Fig. 19 receives an Interest packet for the content, it finds that the content's expected quality at CR_1 is the largest. As a result, it forwards the Interest packet to CR_1 instead of CR_3.

CATT selects any one of the content routers along the downloading path for caching, so it belongs to the heterogeneous and on-path caching categories according to our categorization criteria. In addition, CATT uses an intra-domain cooperation which requires the content sources and the content routers to advertise their content availability within different advertisement scopes.

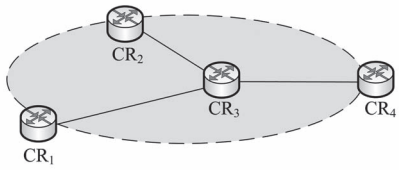
10) Heterogeneous Cache Provision: The caching mechanisms discussed before are classified as homogeneous cache provision where each content router possess the equal cache size. However, Rossi *et al.* [39] propose a heterogeneous caching mechanism, called HSS, to allocate content routers different cache sizes based on certain criteria. Similar to CEE, HSS requires content routers to cache each passing-by Data packet. Therefore, it is an on-path caching mechanism using the uncooperative manner. The difference is that some "important" content routers offer more storage size than the others. One can visualize the decision process of HSS through an illustrative example shown in Fig. 20. Let $G = (V, E)$ be an undirected graph with V content routers and E edges. HSS applies the following six metrics to calculate each content router's cache size.

- Degree Centrality (denoted by DC): $DC(R_n)$ is equal to the number of edges connected to R_n. Their values are listed in Fig. 20.
- Stress Centrality (denoted by SC): $SC(R_n)$ refers to the total number of shortest paths between all other content routers that pass through R_n. $SC(R_n)$ is calculated as follows:

$$SC(R_n) = \sum_{\forall R_i, R_j \in V \setminus R_n} n(R_i, R_j, R_n) \tag{6}$$

where $n(R_i, R_j, R_n)$ is the number of shortest paths between R_i and R_j through R_n.

	DC	SC	BC	CC	GC	EC
CR_1	1	0	0	1/5	1/2	2
CR_2	1	0	0	1/5	1/2	2
CR_3	3	3	1	1/3	1	1
CR_4	1	0	0	1/5	1/2	2



CR: Content Router
DC: Degree Centrality SC: Stress Centrality
BC: Betweenness Centrality CC: Closeness Centrality
GC: Graph Centrality EC: Eccentricity Centrality

Fig. 20. Illustration of HSS.

- Betweenness Centrality (denoted by BC): $BC(R_n)$ is the normalized value of $SC(R_n)$ and reflects how often R_n lies on the shortest paths between all the other content routers in the network. We can calculate the value of the Betweenness Centrality through Eq. (4).
- Closeness Centrality (denoted by CC): $CC(R_n)$ is defined as the hop count of R_n to all the other content routers in the network. It is given as follows:

$$CC(R_n) = \frac{1}{\sum_{\forall R_i \in V \setminus R_n} d(R_n, R_i)} \tag{7}$$

where $d(R_n, R_i)$ is the hop count of the shortest path from R_n to R_i. $CC(R_n)$ represents the sum of R_n's distance to all other content routers of network. It is obvious that the smaller $CC(R_n)$ is, the more concentrated the content router could be in the network.
- Graph Centrality (denoted by GC): $GC(R_n)$ is defined as the hop count of R_n to the farthest content router of the network and given by Eq. (8). The content router with a high $GC(R_n)$ has a short distance from R_n to all other routers.

$$GC(R_n) = \frac{1}{\max_{\forall R_i \in V \setminus R_n} d(R_n, R_i)} \tag{8}$$

- Eccentricity Centrality (denoted by EC): $EC(R_n)$ is equal to the reciprocal of $GC(R_n)$ and reflects how far, at most, it could be from R_n to every other content router in the network. It is calculated as follows:

$$EC(R_n) = \max_{\forall R_i \in V \setminus R_n} d(R_n, R_i) \tag{9}$$

Each content router's cache size, denoted by $Cs(R_n)$, can be calculated through Eq. (10).

$$Cs(R_n) = Cs(T) \frac{X(R_n)}{\sum_{\forall R_i \in V} X(R_i)}, \ \forall R_n \in V \tag{10}$$

where $X(R_n) = [CC(R_n), GC(R_n), DC(R_n), EC(R_n), SC(R_n), BC(R_n)]$ and $Cs(T)$ represents the total storage size

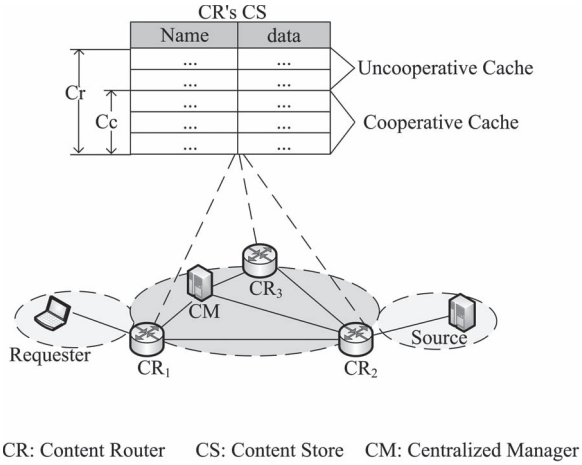


Fig. 21. Illustration of a hierarchical cache mechanism.

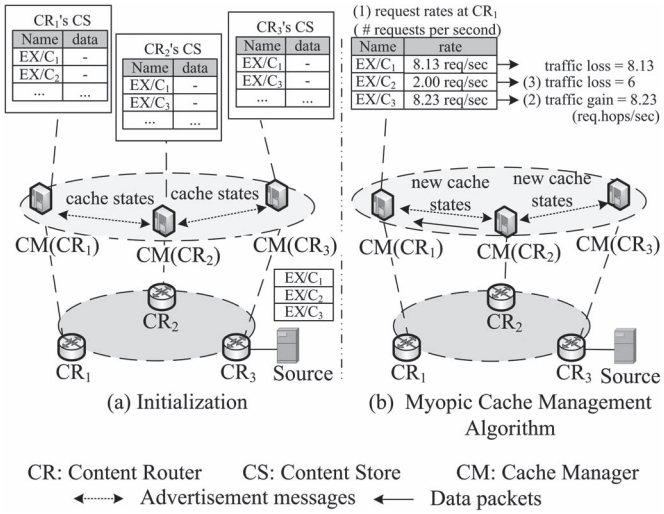


Fig. 22. Illustration of cache configuration scenarios. (a) Initialization; (b) Myopic cache manager algorithm.

of the network. For example, CR_3 has a potential cache size approximately up to 15 GB if $Cs(T)$ reaches 40 GB in Fig. 20, but will only have 10 GB in the case of homogeneous cache provision.

The cache provision mechanism based on some graph-related centrality properties fails to take the traffic distribution and user behaviors into account. To overcome the problem, a new cache size optimization scheme is presented in [45], which continues to allocate more cache size to the "important" content routers, just as the above mentioned scheme does. However, it determines a content router's important property in terms of both traffic distribution and user behaviors. In the cache size optimization scheme, three data sets of each content router, including the number of Interest packets on arrival, the number of Interest packets answered and the number of the cached content replacements are regularly collected for further determining the content router's cache size.

A novel hierarchical cache mechanism for heterogeneous provision is proposed in [46]. First of all, it still assumes that each content router has the same caching capacity. Secondly, a content router's CS is dived into two parts: uncooperative cache and cooperative cache. The cooperative cache is available for all content routers within the domain, while the remaining part is uncooperative and can only be used locally. For example, the cache size of each content router are assumed to be C_r, and the amount of cache size allocated to cooperation is denoted as C_c ($C_c \leqslant C_r$). Finally, by using the replacement algorithm based on frequency, the content router stores the most popular contents firstly in its uncooperative cache, and then stores them in its cooperative cache when the capacity of uncooperative cache gets insufficient. Consider an illustrative example shown in Fig. 21, where there exists a centralized manager which takes charge of maintaining the cache state of the network. Each content router advertises the availability information of cooperative cache to the manager.

11) DCM: DCM proposes that a *Cache Manager* (CM) should be installed to exchange its cache state with other CMs, as depicted in Fig. 22. More importantly, the CM uses a distributed on-line cache management algorithm to decide which cached content to be replaced. Let $G = (V, E)$ be an arbitrary topology which consists of V content routers and E links. The following is a complete list of the algorithm parameters:

- S is the total amount of cached content in a given network G.
- C_r is the average caching capability of content routers (assuming that content routers of the network have equal capacity).
- $r^s_{R_v}$ is the incoming request rate at R_v ($R_v \in V$) for the content s ($s \in S$).
- $h^s_{R_v}$ equals *true* if s is cached by R_v. It is given as follows:

$$h^s_{R_v} = \begin{cases} 1 & \text{if } s \text{ is cached by } R_v. \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

- H is the cache configuration that aims at minimizing the network traffic cost. It can be represented by a binary matrix of size $V \times S$:

$$\begin{bmatrix} h^1_{R_1} & h^2_{R_1} & \dots & h^S_{R_1} \\ h^1_{R_2} & h^2_{R_2} & \dots & h^S_{R_2} \\ \dots & \dots & \dots & \dots \\ h^1_{R_V} & h^2_{R_V} & \dots & h^S_{R_V} \end{bmatrix} \tag{12}$$

The object of DCM is to find an applicable configuration to minimize the network traffic cost. It is formulated as a linear programming:

$$\text{minimize :} \quad T(H) = \sum_{s=1}^{S} r^s_{R_v} d_{R_v R_m} \tag{13}$$

$$\text{subject to :} \quad \sum_{R_v=1}^{V} h^s_{R_v} \geqslant 1 \tag{14}$$

$$\sum_{s=1}^{S} h^s_{R_v} \leqslant C_r \tag{15}$$

where R_m is the nearest cache hosting a replica of a given content s and $d_{R_v R_m}$ is the number of hops (or delay) between R_v and R_m. In particular, Eq. (14) represents the constraint that each cached content has to be stored by at least one content router. The second constraint captures the fact that each content router's caching capacity is limited.

The above issue can be addressed as a knapsack problem [47]. The initial cache configuration is expressed as H_i and its corresponding network traffic cost is labeled as $T(H_i)$. The presented algorithm, called myopic cache management algorithm, is amplified as follows:

- Step 1: $CM(R_v)$ selects a content (e.g., s) and determines whether it has been cached locally.
- Step 2(a): If s has not been cached by R_v yet, $CM(R_v)$ calculates the traffic gain $g_s = T(H_i) - T(H^*)$ once it is cached by R_v, where H^* represents that $h^s_{R_v}$ equals *true* and the other cache configurations remain unchanged.
- Step 2(b): If s has been cached, $CM(R_v)$ calculates the traffic loss $l_s = T(H^*) - T(H_i)$ when it is evicted by R_v, where H^* represents that $h^s_{R_v}$ equals *false* and the other cache configurations remain unchanged.
- Step 3: $CM(R_v)$ repeats steps 1-2 until all of the cached content has been selected. The content with maximum traffic gain is considered as a cached content candidate (e.g., i) and the content with minimum loss is considered as a replacement candidate (e.g., j). If relative gain $R_g = g_i - l_j$ is positive, $CM(R_v)$ fetches the content i and evicts the cached content j. Simultaneously, it sets $h^i_{R_n}$ to be *true* and $h^j_{R_v}$ to be *false*.
- Step 4: $CM(R_v)$ advertises its new cache state to other CMs.
- Step 5: Repeat steps 1–5 in other CMs until there is no replacement that can produce more benefits.

We illustrate the cache configuration scenery of DCM with the help of Fig. 22 and consider a simple network model for NDN. We also assume that the content source publishes contents EX/C_1, EX/C_2 and EX/C_3. Before using the myopic cache management algorithm, the cache states of the network are shown in Fig. 22(a), where CR_1 caches EX/C_1 and EX/C_2. In this case, both CR_2 and CR_3 cache EX/C_1 and EX/C_3. Simultaneously, the cache states of content routers have been exchanged between CMs. Since the service request rate in the literature [40] is an estimation based on historical data, we quote the rate used in the example of the literature [40], as illustrated by (1) in Fig. 22(b). At first, $CM(CR_1)$ selects EX/C_1, EX/C_2 and EX/C_3 respectively, and computes its traffic gain or loss based on whether it is cached locally or not. Moreover, $CM(CR_1)$ chooses the one that has the greatest traffic gain (step 2) and the one of minimum traffic loss (step 3), and calculates the relative gain. As a result, it replaces EX/C_2 with EX/C_3. After setting the new cache configuration, $CM(CR_1)$ advertises the new cache state to other CMs.

It is obvious that DCM focuses more on optimizing the cached content sufficiently to get the best possible network traffic. In the initial cache configuration, CMs can adopt homogeneous or heterogeneous caching mechanism. For example, a content router may cache every incoming Data packet, as it does in CEE. Moreover, CMs fetch and cache the new content which has been cached by other CMs once the algorithm has been implemented. In this case, the replication in DCM is determined by the traffic gain, instead of treating each cached content indistinguishably, so DCM is classified as as heterogenous caching mechanism. Moreover, it is not required for the content to be cached along the delivery path, therefore DCM belongs to the off-path caching category.

E. Summary

In this section we have surveyed a large number of caching mechanisms for ICN architectures. According to the above descriptions, we summarize the optimization means and the advantages of them in Table IV. From this table, we can see that, except ICC and Breadcrumbs, all in-network caching mechanisms aim at improving the cache hit ratio by reducing caching redundancy. In Table V, we further summarize the additional fields, tables and advertisements incurred by different caching mechanisms.

IV. Simulation Experiments

In this section, we evaluate some typical in-network caching mechanisms in terms of the performance metrics described in Section III-A through extensive simulations.

A. Simulation Environments

To investigate the sensitivity of caching mechanisms to network topologies, we use the empirical results from the Tier program [48] to generate two different topologies, whose parameters are shown in Table VI. In the first topology (denoted by T_a), we simulate a three-level network with 32 content routers and the network is composed of a core network (i.e., the Wide Area Network (WAN) in Tiers) and a number of access networks (i.e., Metropolitan Area Networks (MANs) and Local Area Networks (LANs) in Tiers), as illustrated in Fig. 23(a). We assume a scenario of 400 clients and 250 content sources and they are both randomly attached to the access networks. As illustrated in Fig. 23(b), we also simulate a single level network (denoted by T_b) with 30 content routers, where the simulation topology follows the typical Tiers network, and we consider a scenario of 100 clients and 100 content sources and they are both randomly attached to the content routers. In addition, we assume that each content source in different network topologies stores either 120 chunks or 300 chunks and every chunk has a size of 10 KB. The content source advertises the availability of content chunks that it holds, and responses with Data packets when it receives Interest packets. At the same time, clients issue Interest packets for chunks whose popularity follows the Zipf distribution with the Zipf parameter set to 0.95. We also assume that the arrival process of Interest packets for chunks at each content router follows a Poisson process and each content router has the same cache size.

B. Simulation Results

We now present results from extensive simulations to gain some insights into the in-network caching mechanisms. Because there are too many in-network caching mechanisms, it

TABLE IV
SUMMARY OF THE OPTIMIZATION MEANS AND ADVANTAGES

Name	Reducing Caching Redundancy	Improving Cache Availability	Advantages
Prob(p)	Yes	No	caching the popular content, reducing the caching redundancy according to the probability
Breadcrumbs	No	Yes	making the cached content available for off-path requests
Intra-AS Co	Yes	Yes	reducing the caching redundancy within one-hop neighbors, making the cached content available for off-path requests
ICC	No	Yes	making the cached content available for off-path requests, supporting the off-path caching
Probcache	Yes	No	reducing the caching redundancy along path, pushing the content swiftly, caching the popular content by edge content routers
CBC	Yes	No	reducing the caching redundancy along path
LCD and MCD	Yes	No	reducing the caching redundancy along path, caching the popular content by edge content routers
WAVE	Yes	No	reducing the caching redundancy along path, an efficient mechanism for caching large files
CLS	Yes	Yes	reducing the caching redundancy along path, making the cached content available for off-path requests
CINC	Yes	Yes	reducing the caching redundancy within domain, supporting the off-path caching
CPHR	Yes	No	reducing the caching redundancy within domain, making the cached content available for off-path requests
LCC	Yes	Yes	reducing the caching redundancy between neighboring domains, making the cached content available for off-path requests
CATT	Yes	Yes	reducing the caching redundancy within k-hop neighbors, making the cached content available for off-path requests
HSS	Yes	No	allocating more cache size to the "important" content routers
DCM	Yes	Yes	replacing the cached content of minimum traffic loss with the one of maximum traffic gain, making the cached content available for off-path requests

TABLE V
SUMMARY OF THE ADDITIONAL FIELDS, TABLES AND ADVERTISEMENTS IN DIFFERENT CACHING MECHANISMS

Name	Additional Fields	Additional Tables	Additional Cost
CEE, Prob(p) and HSS	null	null	null
Breadcrumbs	null	caching trail	null
Intra-AS Co	null	LCST and ECST	advertising the cache availability
ICC	null	an additional table for recording the cache availability that comes from the peering domain	advertising the cache availability
Probcache	TSI in Interest packets, TSI and TSB in Data packets	null	null
CBC	a field in Interest packets for recording the maximum value	null	null
LCD, MCD and WAVE	a caching flag bit in Data packets	null	null
CLS	a caching flag bit in Data packets	caching trail	null
CINC	a caching flag bit in Interest packets	CRT and CCS	null
CPHR	*Cache Name* field and *Egress Router* field in Interest packets, *Cache Name* field in Data packets	null	null
LCC	a field in Interest packets for recording the FID	an additional routing table	null
CATT	null	PBR which replaces the FIB	advertising the cache availability
DCM	null	a table for recording the cache states shared by other CMs	advertising the cache state

is very difficult to simulate all of them. As a result, we choose CEE [18], Prob(p) [14] to be representatives of non-cooperative caching mechanisms because CEE is the only one that caches Data packets everywhere, and Prob(p) can outperform some cooperative caching mechanisms if the parameter p is appropriately chosen, as will be shown later in this section. Similarly, we choose Probcache [31], LCD [14], and Breadcrumbs [28] to be representatives of cooperative caching mechanisms because,

TABLE VI
SIMULATION PARAMETERS

Parameters	Values	
Network Topologies	T_a	T_b
Number of Content Routers	32	30
Number of Domains	10	1
Number of Clients	400	100
Number of Sources	250	100
Number of Chunks	30000	
Average Size of Chunks	10KB	

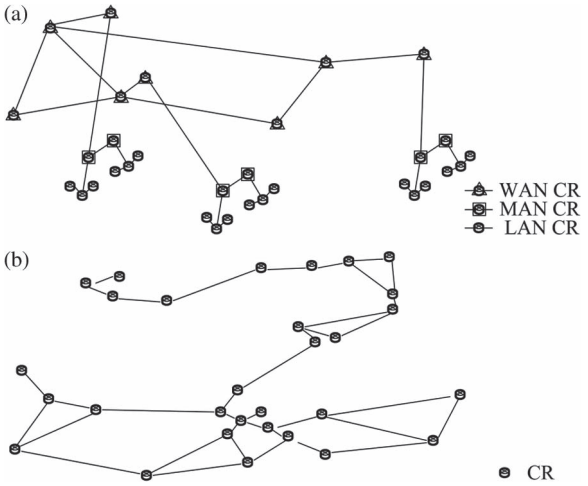


Fig. 23. Network topologies for simulations.

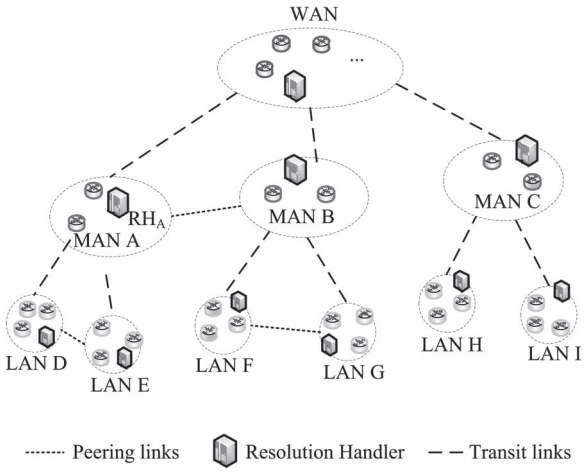


Fig. 24. The relationship between domains in simulations.

to the best of our knowledge, they appeared in the literature most frequently. We also simulate ICC and compare it with the original caching approach used in DONA to evaluate the benefits and drawbacks of inter-domain cooperative caching. For clarity, we divide the results into three scenarios: non-cooperative caching, cooperative caching, and inter-domain cooperative caching. In all the following figures, the x-axis represents the cache size, which is expressed as a proportion of total content bytes in simulations and ranges from 1% to 3%.

1) Scenario 1—Non-Cooperative Caching Approaches: Fig. 25 shows the simulation results from the two different topologies when the LRU replacement algorithm is used. From Fig. 25(a) and (c), we observe that the hit ratio increases with the increase of the cache size. This is because a content router with a larger cache size can cache more contents, thus leading to a higher chance to answer Interest packets. At the same time, we also observe that the cache hit distance decreases with the increase of cache size, because more Interest packets are answered by the content routers that are closer rather than the content source. More importantly, we observe that in these two topologies, Prob(p) has higher cache hit ratio and shorter cache hit distance than CEE because Prob(p) eliminates some caching redundancy, as discussed in Section IV-B. We can also observe that, for Prob(p), the smaller the caching probability is, the higher is the cache hit rate and the lower is the cache hit distance. This is because more caching redundancy can be eliminated when the caching probability p decreases.

We also investigated the impact of the cache replacement algorithms. With the LRU replacement algorithm, a content router orders cached contents based on the recency of use. When a cache replacement is required, the LRU algorithm removes from the cache the content that has not been referenced for the longest period of time. On the other hand, the LFU replacement algorithm is a frequency-based policy that tries to keep popular contents in the cache. When a cache replacement is required, the LFU algorithm removes the cached content with the least access count. Fig. 26 compares the caching performance of the LFU algorithm and the LRU algorithm. We observe that, for both CEE and Prob(p), the LFU replacement algorithm performs better than the LRU replacement algorithm, in terms of both cache hit ratio and cache hit distance. Considering that LRU is the most basic cache replacement algorithm, we only present results for the LRU algorithm in the rest of the paper.

2) Scenario 2—Cooperative Caching Approaches: Fig. 27 compares the performance of the chosen cooperative caching approaches, namely LCD, Probcache, and Breadcrumbs. From this figure, we observe that LCD has the highest cache hit ratio among them because, as discussed in Section IV-C, LCD eliminates more caching redundancy by caching the most popular contents close to users. On the other hand, we observe that the hit distance of Breadcrumbs is longer than those of the other two mechanisms because, as discussed in Section IV-C, the downstream content routers in Breadcrumbs may have deleted

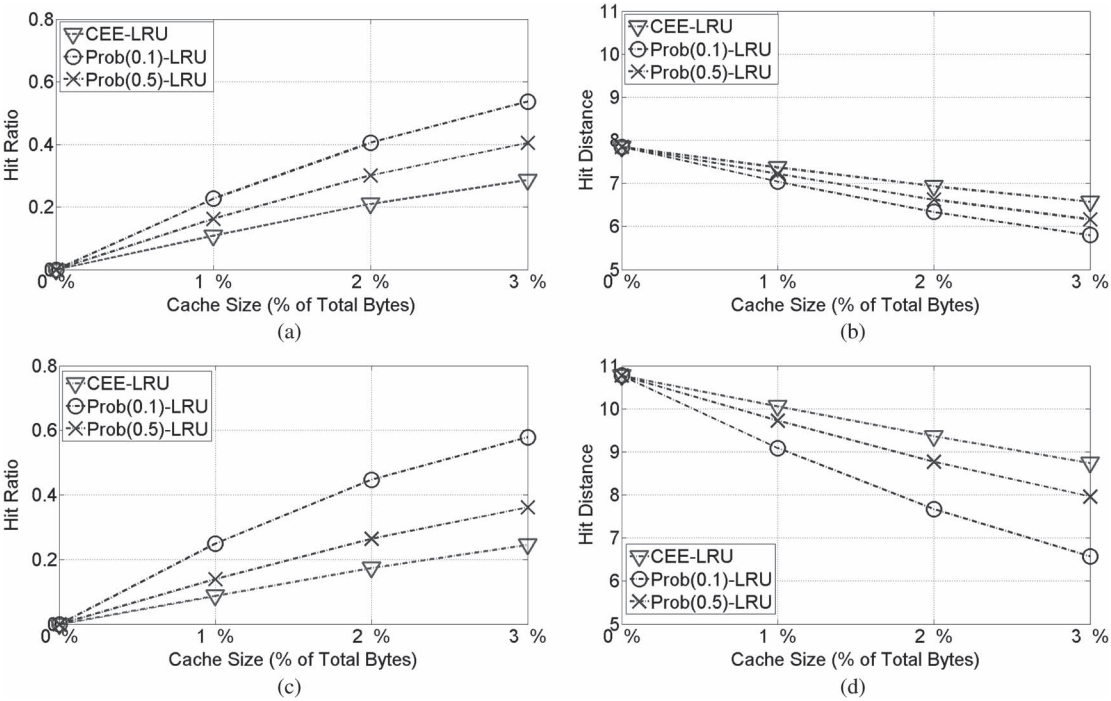


Fig. 25. Comparison of CEE and Prob(p) when LRU is used. (a) Hit ratio optimization on T_a; (b) hit distance optimization on T_a; (c) hit ratio optimization on T_b; (d) hit distance optimization on T_b.

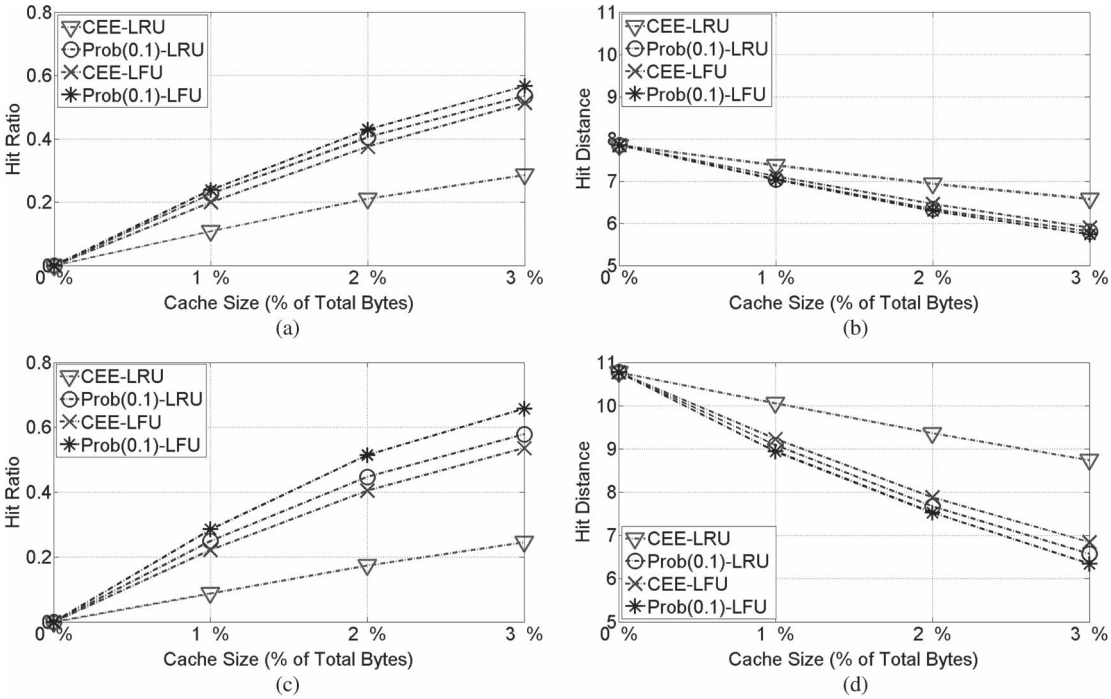


Fig. 26. Comparison of the LRU and LFU algorithms. (a) Hit ratio optimization on T_a; (b) hit distance optimization on T_a; (c) hit ratio optimization on T_b; (d) hit distance optimization on T_b.

the cached content, which in turn causes the requests to be re-forwarded to the sources. Note that, although LCD has the highest cache hit ratio, its cache hit distance is not the least. This can be explained as follows. Although LCD is capable of caching the frequently accessed contents, but it cannot push the popular contents to the network edge quickly, as discussed in Section IV-D. Furthermore, the limited cache size causes some of the popular contents to be replaced in the diffusion process.

Comparing Figs. 26 and 27, we can find that Breadcrumbs has higher cache hit ratio than CEE. For example, at cache size 3%, the cache hit ratios of CEE and Breadcrumbs are 28.7% and

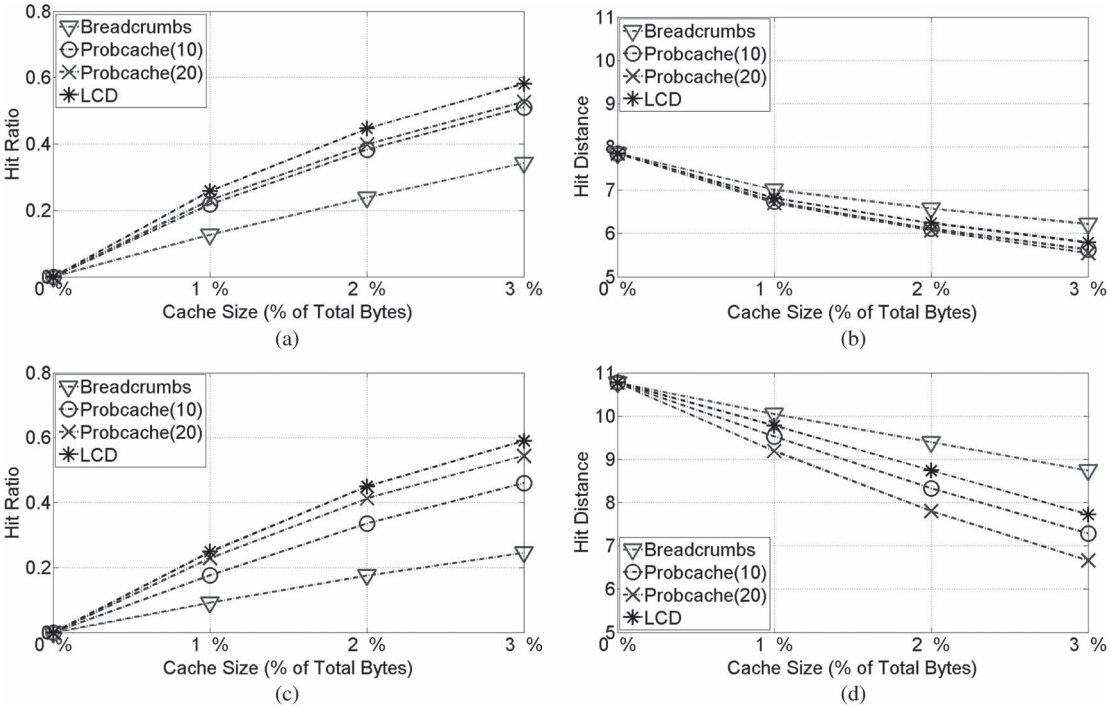


Fig. 27. Comparison of Probcache, LCD and Breadcrumbs. (a) Hit ratio optimization on T_a; (b) hit distance optimization on T_a; (c) hit ratio optimization on T_b; (d) hit distance optimization on T_b.

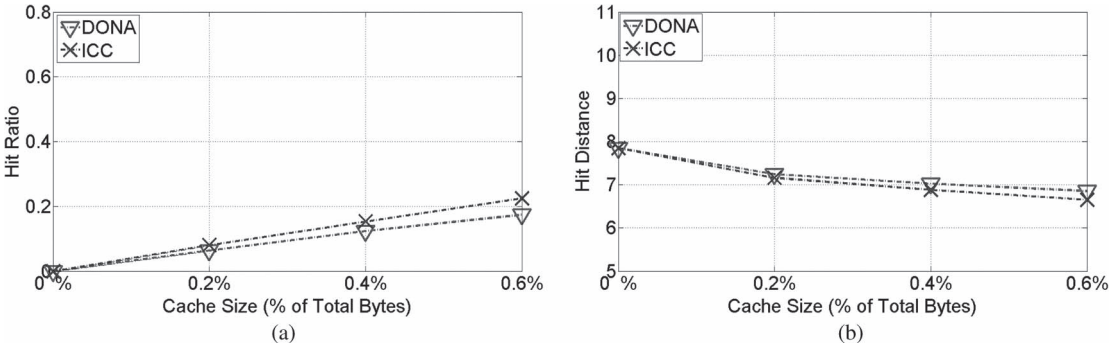


Fig. 28. The effect of inter-domain cooperation. (a) Hit ratio optimization on T_a; (b) hit distance optimization on T_a.

34.2%, respectively, in the topology T_a. Similarly, the average hop counts of cache hit distance of CEE and Breadcrumbs are 6.58 and 6.22, respectively. So, Breadcrumbs performs better than CEE in our simulations. The reason behind is that a cached Data packet is available locally in CEE but is available along the path from the content source to the requester in Breadcrumbs. That is, Breadcrumbs makes the cached content available for off-path requests.

Since Breadcrumbs has the worst cache hit ratio among the simulated cooperative caching mechanisms, it seems that cooperative caching mechanisms perform better than non-cooperative caching mechanisms. Unfortunately, this is not the case. Indeed, by comparing the cache hit ratio of Breadcrumbs and that of Prob(p) when $p = 0.1$ at cache size 3%, we can find that the cache hit ratios of Breadcrumbs and Prob(0.1) are 34.2% and 53.6%, respectively, in the topology T_a. The reason is because when $p = 0.1$, Prob(p) eliminates more caching redundancy than Breadcrumbs.

3) Scenario 3— The Effect of Inter-Domain Cooperation: To investigate the effect of inter-domain cooperation, we also simulated ICC and the default caching mechanism in DONA (denoted by DONA in Fig. 28). In the latter case, each RH caches every incoming DATA packet but does not advertise its availability information to other domains. On the other hand, as described in Section IV-C, the RH in ICC advertises the availability of cached content to its peering domain. Fig. 28 compares the cache hit ratio and the cache hit distance of ICC and DONA when the topology shown in Fig. 24 is used. From Fig. 28, we observe that ICC is able to increase the cache hit ratio by 5% compared to DONA when the cache size equals to 0.6%. Note that the cache size is limited in the simulations because each domain is assumed to have only one RH as a cache-equipped device. This implies that inter-domain cooperation can increase the cache hit ratio.

However, this comes at increased operational cost. In ICC, the frequency of replacement determines the number of

TABLE VII
THE NUMBER OF ADVERTISEMENT MESSAGES SENT BY THE RH IN DOMAIN A AS THE NUMBER OF FIND MESSAGES AMOUNTS TO 30000

Cache Size (% of Total bytes)	The Number of Advertisement Messages
0.2%	8000
0.4%	7706
0.6%	7371

advertisement messages. Table VII shows the number of advertisement messages sent by RH_A. As the relative cache size becomes 0.6% and the number of requests in the simulations amounts to 30,000, the number of advertisement messages sent by RH_A reaches 7,370. Note that the number of advertisement messages will decrease when the cache size increases because the greater the cache is, the more FIND packets it can respond to, thus reducing the replacement frequency (i.e., the number of advertisement messages).

C. Discussion

In this section, we have compared the caching performance of several typical mechanisms by conducting extensive simulations. Our numerical results provide evidence that eliminating (or reducing) the caching redundancy is able to improve the cache hit ratio and reduce the cache hit distance. Compared with the non-cooperative caching mechanism, the cooperative caching mechanism generally (not always) achieves a higher cache hit ratio for being able to reduce the caching redundancy efficiently. Besides, advertising the cache availability can optimize the cache hit ratio and cache hit distance, but incurring a significant cost.

VI. REMAINING RESEARCH CHALLENGES

Although the networking community has made considerable achievements in in-network caching mechanisms, there are many remaining research challenges.

A. The Tradeoff Between Gain and Cost

Some of the currently available in-network caching mechanisms use cache state advertisement to reduce caching redundancy and to improve cache availability. However, the volatility of cached content results in frequent advertisement, thus limiting the benefits that an advertisement mechanism can bring about. Therefore, how to achieve the balance between this gain and advertisement cost is an interesting topic. Meanwhile, each ICN approach comes about with its own specific characteristic that impacts the choice of the most appropriate caching mechanism. For instance, the name-resolution-based approaches [49], [50] adopt name resolution services [51], hence the caching mechanisms proposed for different ICN approaches need to be eligible to support their features.

B. In-Network Caching in Mobile Networks

Mobile operators have been impacted by the surge in mobile data traffic on their networks and look forward to deploying an effective in-network caching mechanism to reduce network traffic and user access latency, as in wired environments. However, users in mobile networks roam from place to place. So, how to take user mobility into consideration and design appropriate in-network caching mechanisms is of paramount importance, especially in the coming mobile Internet era.

C. In-Network Caching in Software-Defined Networking

Software Defined Networking (SDN) is a new network paradigm, in which the data plane and the forwarding plane are separated [52]. An important feature of SDN is that the logical centralized controller knows all states of network elements, which makes it possible to manage network resources efficiently. Because of this, many researchers have tried to integrate SDN and ICN into a coherent future Internet architecture [53], [54]. In that case, should in-network caching be realized in a cooperative way, or in a distributed manner where network devices make caching decisions independently, as in many existing in-network caching mechanisms? In addition, if in a centralized way, how to guarantee the robustness of in-network caching mechanisms in case of the controller failures? These are also challenges for the networking community to answer.

D. In-Network Caching in the Internet of Things

The Internet of Things (IoT) is designed to connect billions of objects to the Internet and can leverage the ICN features to obtain more benefits. For instance, smart grids use in-network caching to back up some valuable contents. In addition, considering the resource constraints in IoT, such as limited wireless bandwidth, power supply and caching capacity, it is worthy of saving wireless bandwidth and battery power by avoiding unnecessary caching and storing some valuable contents at the IoT devices. In this context, the need for an effective caching mechanism becomes more critical. Furthermore, due to the IoT features, there are some exceptions that in-network caching fails to provide an ideal benefit. For example, when the latest value recorded by a sensor is requested, the returned data may be outdated because the corresponding values have been updated. In that case, in-network caching may not provide the desired benefits.

E. Deployments of In-Network Caching

As stated previously in Section I, in-networking caching is an important feature of almost all ICN architectures. Accordingly, the deployability of ICN architectures heavily depends on that of the in-network caching mechanisms [55]. Unfortunately, many existing caching mechanisms fail to consider the problem

of how to incrementally deploy in-network caching in ICN. To address this issue, the first step should to gain some insights on existing in-network caching mechanisms by deploying them in real networks (or testbeds). Accordingly, how to overcome the limited size of testbeds for ICN architectures is another challenge.

VII. Conclusion

In-network caching is an important characteristic of ICN and many in-network caching mechanisms have been proposed in recent years. In this paper, we have firstly described some performance measures used to evaluate the in-network caching mechanisms and presented some criteria for categorizing the in-network caching mechanisms. Then, we have made a comprehensive overview of the recently proposed caching mechanisms by describing them in detail, discussing their possible advantages and disadvantages, and pointing out their similarities and differences. We have also compared the caching performance of several typical mechanisms through extensive simulations. In summary, we have found that eliminating (or reducing) the caching redundancy is an effective way to improve the cache hit ratio, reduce the user access latency and the traffic sent to servers. In addition, cooperative caching mechanisms generally (not always) have better cache hit ratio than non-cooperative caching mechanisms, because they often have less caching redundancy. However, this comes at the cost of complexity such as additional fields for carrying caching information or notification of cache availability. Therefore, how to design a caching mechanism that achieves high hit ratio, short hit distance and low cost is still an interesting and challenging issue. This together with the challenges described in Section VI are worthy of further investigation for ICN to have big impact on our society.

Acknowledgment

We thank the anonymous reviewers for their invaluable comments that improve the paper.

References

[1] Cisco, Visual networking index: Forecast and methodology, 2013-2018, Jun. 2014, White Paper. [Online]. Available:http://www.cisco.com/go/vni

[2] G. Xylomenos *et al.*, "A survey of information-centric networking research," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 1024–1049, Jul. 2013.

[3] M. Gritter and D. R. Cheriton, "An architecture for content routing support in the Internet," in *Proc. USENIX Symp. Internet Technol. Syst.*, San Francisco, CA, USA, 2001, pp. 37–48.

[4] Stanford University TRIAD Project. [Online]. Available: http://wwwdsg.stanford.edu/triad/

[5] T. Koponen *et al.*, "A data-oriented (and beyond) network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 181–192, Oct. 2007.

[6] Content Centric Networking Project. [Online]. Available: http://www.ccnx.org/

[7] NSF Named Data Networking Project. [Online]. Available: http://www.named-data.net/

[8] FP7 PSIRP Project. [Online]. Available: http://www.psirp.org/

[9] FP7 PURSUIT project. [Online]. Available: http://www.fp7-pursuit.eu/PursuitWeb/

[10] FP7 4WARD Project. [Online]. Available: http://www.4ward-project.eu/

[11] FP7 SAIL Project. [Online]. Available: http://www.sail-project.eu/

[12] FP7 COMET Project. [Online]. Available: http://www.comet-project.org/

[13] NSF Mobility First Project. [Online]. Available: http://mobilityfirst.winlab.rutgers.edu/

[14] 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.

[15] S. Wang, J. Bi, and J. Wu, "Collaborative caching based on hash-routing for information-centric networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 535–536, Oct. 2013.

[16] G. Tyson *et al.*, "A trace-driven analysis of caching in content-centric networks," in *Proc. IEEE 21st Int. Conf. Comput. Commun. Netw.*, Munich, Germany, 2012, pp. 1–7.

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

[18] V. Jacobson *et al.*, "Networking named content," in *Proc. ACM CoNEXT Conf.*, Rome, Italy, 2009, pp. 1–12.

[19] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou, "Modelling and evaluation of ccn-caching trees," in *Proc. 10th Int. IFIP TC 6 Netw. Conf.*, Valencia, Spain, 2011, pp. 78–91.

[20] L. Zhang *et al.*, "Named data networking (NDN) project," Palo Alto Res. Center, Palo Alto, CA, USA, Tech. Rep. NDN-0001, Oct. 2010.

[21] G. Rossini and D. Rossi, "A dive into the caching performance of content centric networking," in *Proc. IEEE 17th Int. Workshop CAMAD*, Barcelona, Spain, 2012, pp. 105–109.

[22] L. Muscariello, G. Carofiglio, and M. Gallo, "Bandwidth and storage sharing performance in information centric networking," in *Proc. 1st ACM SIGCOMM Workshop ICN*, Toronto, ON, Canada, 2011, pp. 26–31.

[23] G. Bianchi, A. Detti, A. Caponi, and N. Blefari Melazzi, "Check before storing: What is the performance price of content integrity verification in lru caching?" *ACM SIGCOMM Comput. Comm. Rev.*, vol. 43, no. 3, pp. 59–67, Jul. 2013.

[24] K. Katsaros, G. Xylomenos, and G. C. Polyzos, "Multicache: An overlay architecture for information-centric networking," *Comput. Netw.*, vol. 55, no. 4, pp. 936–947, Mar. 2011.

[25] K. Katsaros, G. Xylomenos, and G. C. Polyzos, "A hybrid overlay multicast and caching scheme for information-centric networking," in *Proc. IEEE 29th Int. Conf. Comput. Commun. Workshops*, San Diego, CA, USA, 2010, pp. 1–6.

[26] S. Arianfar, P. Nikander, and J. Ott, "On content-centric router design and implications," in *Proc. Re-Archit. Internet Workshop*, Philadelphia, PA, USA, 2010, p. 5.

[27] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Comput. Surveys*, vol. 35, no. 4, pp. 374–398, Dec. 2003.

[28] E. J. Rosensweig and J. Kurose, "Breadcrumbs: Efficient, best-effort content location in cache networks," in *Proc. IEEE 28th Int. Conf. Comput. Commun.*, Rio de Janeiro, Brazil, 2009, pp. 2631–2635.

[29] J. M. Wang, J. Zhang, and B. Bensaou, "Intra-as cooperative caching for content-centric networks," in *Proc. 3rd ACM SIGCOMM Workshop ICN*, Hong Kong, 2013, pp. 61–66.

[30] J. Rajahalme, M. Särelä, P. Nikander, and S. Tarkoma, "Incentive-compatible caching and peering in data-oriented networks," in *Proc. ACM CoNEXT Conf.*, Madrid, Spain, 2008, pp. 62:1–62:6.

[31] 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, Dec. 2013.

[32] I. Psaras, W. K. Chai, and G. Pavlou, "Probabilistic in-network caching for information-centric networks," in *Proc. 2nd ACM SIGCOMM Workshop ICN*, Helsinki, Finland, 2012, pp. 55–60.

[33] W. K. Chai, D. He, I. Psaras, and G. Pavlou, "Cache less for more in information-centric networks," in *Proc. 11th Int. IFIP TC 6 Netw. Conf.*, Prague, Czech Republic, 2012, pp. 27–40.

[34] K. Cho *et al.*, "Wave: Popularity-based and collaborative in-network caching for content-oriented networks," in *Proc. IEEE 31st Int. Conf. Comput. Commun. Workshops*, Orlando, FL, USA, 2012, pp. 316–321.

[35] Y. Li, T. Lin, H. Tang, and P. Sun, "A chunk caching location and searching scheme in content centric networking," in *Proc. IEEE Int. Conf. Commun.*, Ottawa, ON, Canada, 2012, pp. 2655–2659.

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

[37] S. Saha, A. Lukyanenko, and A. Yla-Jaaski, "Cooperative caching through routing control in information-centric networks," in *Proc. IEEE 32nd Int. Conf. Comput. Commun.*, Turin, Italy, 2013, pp. 100–104.

[38] S. Eum, K. Nakauchi, M. Murata, Y. Shoji, and N. Nishinaga, "Catt: Potential based routing with content caching for ICN," in *Proc. 2nd ACM SIGCOMM Workshop ICN*, Helsinki, Finland, 2012, pp. 49–54.
[39] D. Rossi and G. Rossini, "On sizing ccn content stores by exploiting topological information," in *Proc. IEEE 31st Int. Conf. Comput. Commun. Workshops*, Orlando, FL, USA, 2012, pp. 280–285.
[40] V. Sourlas, L. Gkatzikis, P. Flegkas, and L. Tassiulas, "Distributed cache management in information-centric networks," *IEEE Trans. Netw. Serv. Manage.*, vol. 10, no. 3, pp. 286–299, May 2013.
[41] A. Dan and D. Towsley, "An approximate analysis of the LRU and FIFO buffer replacement schemes," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 18, no. 1, May 1990.
[42] F. Kuhn and R. Wattenhofer, "Constant-time distributed dominating set approximation," *Distrib. Comput.*, vol. 17, no. 4, pp. 303–310, May 2005.
[43] A. K. Parekh, "Analysis of a greedy heuristic for finding small dominating sets in graphs," *Inf. Process. Lett.*, vol. 39, no. 5, pp. 237–240, Sep. 1991.
[44] P. Godfrey, I. Ganichev, S. Shenker, and I. Stoica, "Pathlet routing," *ACM SIGCOMM Comput. Comm. Rev.*, vol. 39, no. 4, pp. 111–122, Oct. 2009.
[45] Y. Xu *et al.*, "A novel cache size optimization scheme based on manifold learning in content centric networking," *J. Netw. Comput. Appl.*, vol. 37, pp. 273–281, Jan. 2014.
[46] Y. Li, H. Xie, Y. Wen, and Z.-L. Zhang, "Coordinating in-network caching in content-centric networks: Model and analysis," in *Proc. IEEE 33rd Int. Conf. Distrib. Comput. Syst.*, Philadelphia, PA, USA, 2013, pp. 62–72.
[47] V. V. Vazirani, *Approximation Algorithms*. New York, NY, USA: Springer-Verlag, 2001.
[48] K. L. Calvert, M. B. Doar, and E. W. Zegura, "Modeling Internet topology," *IEEE Commun. Mag.*, vol. 35, no. 6, pp. 160–163, Jun. 1997.
[49] 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.
[50] H. Luo *et al.*, "Color: An information-centric Internet architecture for innovations," *IEEE Netw.*, vol. 28, no. 3, pp. 4–10, May/Jun. 2014.
[51] G. Xylomenos, X. Vasilakos, C. Tsilopoulos, V. A. Siris, and G. C. Polyzos, "Caching and mobility support in a publish-subscribe Internet architecture," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 52–58, Jul. 2012.
[52] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 27–51, Jun. 2014.
[53] J. Wang *et al.*, "Sd-ICN: An interoperable deployment framework for software-defined information-centric networks," in *Proc. IEEE 33rd Int. Conf. Comput. Commun. Workshops*, Toronto, ON, Canada, 2014, pp. 149–150.
[54] H. Luo, J. Cui, Z. Chen, M. Jin, and H. Zhang, "Efficient integration of software defined networking and information-centric networking with color," in *Proc. IEEE Global Commun. Conf.*, Austin, TX, USA, 2014, pp. 1962–1967.
[55] S. K. Fayazbakhsh *et al.*, "Less pain, most of the gain: Incrementally deployable icn," *ACM SIGCOMM Comput. Comm. Rev.*, vol. 43, no. 4, pp. 147–158, Oct. 2013.

Meng Zhang was born in Chongqing, China, in 1988. He received the B.E. degree in communication and information systems from the Beijing Jiaotong University, Beijing, China, in 2010. He is currently pursuing the Ph.D. degree at the School of Electronic and Information Engineering, Beijing Jiaotong University. He has participated in two National Basic Research Programs of China ("973 Program"), including the Smart Identifier Networks which aims at developing a clean-slate Future Internet architecture.

Hongbin Luo received the B.S. degree from Beijing University of Aeronautics and Astronautics in 1999 and the M.S. (with honors) and Ph.D. degrees in communications and information science from University of Electronic Science and Technology of China (UESTC), in June 2004 and March 2007, respectively.

In June 2007, he joined the School of Electronic and Information Engineering, Beijing Jiaotong University, where he is a Professor. From September 2009 to September 2010, he was a Visiting Scholar at the Department of Computer Science, Purdue University. He has authored more than 50 peer-reviewed papers in leading journals (such as IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS) and conference proceedings. In 2014, he won the National Science Fund for Excellent Young Scholars from the National Natural Science Foundation of China (NSFC). His research interests are in the wide areas of network technologies including routing, Internet architecture, and optical networking.

Hongke Zhang received the M.S. and Ph.D. degrees in electrical and communication systems from the University of Electronic Science and Technology of China (formerly known as Chengdu Institute of Radio Engineering) in 1988 and 1992, respectively.

From September 1992 to June 1994, he was a Postdoctoral Research Associate at Beijing Jiaotong University (formerly known as Northern Jiaotong University). In July 1994, he joined Beijing Jiaotong University, where he is a Professor. He has published more than 100 research papers in the areas of communications, computer networks, and information theory. He is the author of eight books written in Chinese and the holder of more than 30 patents.

Dr. Zhang received the Zan Tianyou Science and Technology Improvement Award in 2001, the Mao Yisheng Science and Technology Improvement Award in 2003, the First Class Science and Technology Improvement Award of the Beijing government in 2005, and other various awards. He is now the Chief Scientist of a National Basic Research Program ("973 Program").