In-Network Cache Management and Resource Allocation for Information-Centric Networks

Ioannis Psaras, *Member, IEEE*, Wei Koong Chai, *Member, IEEE*, and George Pavlou, *Senior Member, IEEE*

Abstract—We introduce the concept of *resource management* for in-network caching environments. We argue that in *Information-Centric Networking* environments, deterministically caching content messages at predefined places along the content delivery path results in unfair and inefficient *content multiplexing* between different content flows, as well as in significant *caching redundancy*. Instead, allocating resources along the path according to content flow characteristics results in better use of network resources and therefore, higher overall performance. The design principles of our proposed in-network caching scheme, which we call *ProbCache*, target these two outcomes, namely *reduction of caching redundancy* and *fair content flow multiplexing* along the delivery path. In particular, $ProbCache$ approximates the *caching capability* of a path and caches contents probabilistically to: 1) leave caching space for other flows sharing (part of) the same path, and 2) fairly multiplex contents in caches along the path from the server to the client. We elaborate on the content multiplexing fairness of $ProbCache$ and find that it sometimes behaves in favor of content flows connected far away from the source, that is, it gives higher priority to flows travelling longer paths, leaving little space to shorter-path flows. We introduce an enhanced version of the main algorithm that guarantees fair behavior to all participating content flows. We evaluate the proposed schemes in both homogeneous and heterogeneous cache size environments and formulate a framework for resource allocation in in-network caching environments. The proposed probabilistic approach to in-network caching exhibits ideal performance both in terms of *network resource utilization* and in terms of *resource allocation fairness* among competing content flows. Finally, and in contrast to the expected behavior, we find that the efficient design of $ProbCache$ results in fast convergence to caching of popular content items.

Index Terms—Information-centric networks, in-network caching, content multiplexing, Cache capacity

1 Introduction

INFORMATION- or Content-Centric Networks (ICN/CCN) have been recently proposed as an alternative to the traditional host-to-host communication paradigm [1], [2]. One fundamental property of ICNs is direct naming of individual content objects, instead of their respective end-host machines [3], [4]. In turn, request routing has to be directly associated with content names, making routing *symmetry* another property of the majority of ICN routing approaches (i.e., forward and return paths have to be the same).

Naming content objects and routing to those objects directly, instead of the machines that host them, gives the opportunity to identify contents as they travel from source to destination [1], [2], [3]. In turn, given that the network transfers *named* objects (instead of unidentifiable *data containers*, i.e., IP packets), these objects can be cached *in the network* and be forwarded to subsequent users interested in the same content [5], [6].

In-network caching has therefore emerged as a distinct research field in the context of *Information-Centric Networks*. In-network caching exhibits fundamental differences from overlay Web-caching [7], [8], or hierarchical and co-operative caching approaches [9], [10] and poses new challenges [11], [12]. For instance, past research considered mainly caching of whole files (with a few exceptions, [13]) as well as administration of their placement [14] and location [15] by centralized entities, e.g., DNS and HTTP redirection. Centralized administration of content placement gives the opportunity to control and manage network resources better at the cost of: 1) increased communication overhead to update the content location database, and 2) reduced flexibility in terms of available cache locations.

In contrast, Information-Centric Networking enables caching of addressable content *chunks* [2], [13] in *any* cache-equipped network device and replacement of cached chunks at line-speed [16], [17]. Although, this operation increases the availability of cache locations [18], it also introduces new problems. In particular, line-speed operation renders prohibitive the process of updating logically-centralized content location databases with the exact location of cached contents. This operation in turn leads to decentralized, location-independent management of contents and caches, which alters many of the basic features of past overlay caching techniques, e.g., content-to-cache allocation [14], while it invalidates the applicability of some others, e.g., content placement based on fixed overlay topologies of caches and servers [15].

In this paper, we address the problem of cache management operations that have to be adjusted to fit in a completely decentralized and uncoordinated environment. We

• *The authors are with the Department of Electrical and Electronic Engineering, University College London. E-mail: {i.psaras, w.chai, g.pavlou}@ucl.ac.uk.*

Manuscript received 28 Oct. 2012; revised 19 Nov. 2013, and accepted 21 Nov. 2013. Date of publication 12 Dec. 2013; date of current version 15 Oct. 2014.
Recommended for acceptance by M.E. Acacio.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2013.304

TABLE 1
Link Speeds and Related Caching Properties

LINK NAME	LINK SPEED	1-SEC OF TRAFFIC	SECS OF TRAFFIC IN A 10GB CACHE
OC-24	1,2 Gbps	$\sim$ 0.15 GBs	$\sim$ 64 secs
OC-48	2,4 Gbps	$\sim$ 0.31 GBs	$\sim$ 32 secs
OC-192	9,9 Gbps	$\sim$ 1.25 GBs	$\sim$ 4 secs
OC-768	39,8 Gbps	$\sim$ 5 GBs	$\sim$ 2 secs
OC-1536	79,6 Gbps	$\sim$ 10 GBs	$\sim$ 1 sec
OC-3072	159,2 Gbps	$\sim$ 20 GBs	$\sim$ 0.5 secs

focus on the allocation of the available cache capacity along a *path* of caching entities among different content flows. We consider each path of caching entities as a pool of caching resources and try to find optimal ways of distributing content in these caches. Subsequently, our goal is to reduce caching redundancy and make more efficient use of available cache resources, in order to increase user-perceived quality.

To achieve our goal, we approximate the *caching capability* of a given path *per unit time* (Section 2) and we design *ProbCache*, a *probabilistic algorithm* for distributed content caching and fair content multiplexing along a path of caches (Section 3). In particular, *ProbCache* allocates caching space to content flows based on the number of hops from source to destination. That is, flows connected close to the content source will be given priority to cache in the available caches along their short route, over content flows that travel further away and which have the opportunity to cache their contents in other nodes along the path. Content multiplexing fairness is therefore, associated with the amount of caching resources along a path of caches that a specific content flow utilizes.

ProbCache was initially introduced in [19]. In the present study, we elaborate on the content flow multiplexing fairness of *ProbCache* and find that although it significantly improves the overall network performance (e.g., in terms of cache hits), it sometimes fails to multiplex content flows in a fair manner (Section 4). Based on these observations, we apply modifications to the original design of *ProbCache* in Section 4.2. We propose an enhanced version of the algorithm (called *ProbCache+*) in Section 4.2, which together with our analytical modeling constitute the main contributions of this paper.

We evaluate the performance of both versions of *ProbCache* under several conditions in Section 5. We compare *ProbCache* against well-known caching approaches, such as universal caching and Leave Copy Down (LCD) [20]. Our results suggest that there is indeed a lot of space for resource management optimization of in-network caching policies, given that appropriate resource allocation rules are in place (Section 5). We show that careful content flow multiplexing in caches can achieve up to 20 percent more cache hits in case of small scale flash crowd events and an average of 11-13 percent under normal conditions. This translates to an order of magnitude reduction in terms of cache evictions, which in turn means less computation load and longer cache times for individual contents. We define the *Content Multiplexing Fairness Index* to better capture the resource allocation properties of the protocols and show that *ProbCache* exhibits desirable properties in that respect as well.

This paper includes the following companion supplementary sections, which are available in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.304: 1) the extension of the basic algorithm [19] for heterogeneous cache environments (Section 8), 2) the analytical modeling of both the basic and the enhanced version of the algorithm (Sections 9 and 10), 3) one extra evaluation scenario with variable cache sizes (Section 11) and 4) a comprehensive survey of related works in the area (in Section 12).

We use the terms ''router'', ''cache'', and ''node'' interchangeably to refer to cache-enabled network devices [16]; it should be noted that our approach does not require every router to be cache-enabled, but it will work in hybrid architectures as well. Furthermore, we refer to content ''messages'' and ''chunks'' interchangeably to refer to the *cacheable unit*, which is not necessarily of similar size to an IP packet. In fact, we leave open the actual size of the cacheable unit which is yet to be defined by the ICN research community. We highlight that the concepts and algorithms proposed in this paper are *cache unit-* as well as *architecture-agnostic* and would directly apply to those ICN environments where symmetric routing is used, e.g., [2], [21], while it will need slight modifications to fit to asymmetric routing architectures, e.g., [22], [23], [24].

2 System Model and Assumptions

We argue that *in-network cache management has to take into account the approximate cache capacity of the path of caches and the estimated amount of traffic that these caches serve per unit time, in order to make decisions on whether to cache incoming contents or not*. In Section 2.1, we make assumptions to approximate the cache capacity of a given path and in Section 2.2, we present our design principles.

2.1 Assumptions on Caching Technologies

By definition, *caching* is different to *storage*, both in networks and in computer systems, in that caching keeps contents stored *for a specific amount of time and not indefinitely*, as in storage. Therefore, the size of a cache is a relative factor, which cannot stand on its own, but instead has to be linked to *the amount of time that a given content is cached for*. We therefore, associate the cache size with the traffic that the corresponding router serves per second. Our cache size unit is the *number of seconds worth of traffic cached* in a given router and depends on the speed of the outgoing links of the router in question.

One important question then is: ''*For how long can we afford to cache contents in a given router*?''. Furthermore, given that we are concerned with *paths* of caches and not with single-caches only, another important question is: ''*For how long do we need to cache contents in a given path in order to minimize redundant traffic and maximize gain*?''. Our reasoning for answering these questions is as follows:

- Today's memory access technologies guarantee *line-speed* access to DRAM or RLDRAM chips of up to 10 GBytes at a reasonable price [16]. This means that a 5 GByte-long cache behind a 40 Gbps link can safely be assumed to hold contents for one second (see also Table 1) [17]. *Without loss of generality, we assume that each cache along a path has sufficient*

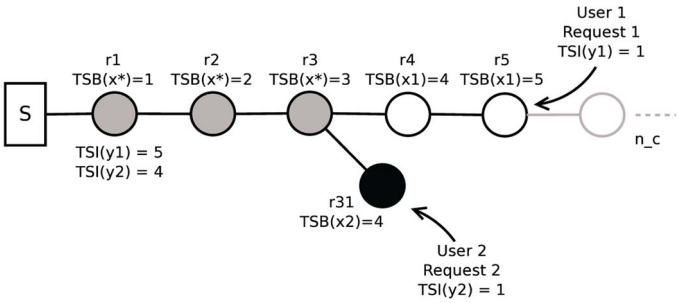

Fig. 1. Design topology.

TABLE 2
Model Notation

SYMBOL	MEANING
n	Number of caches on the path
N_i	Cache size of router r_i: holds 1-sec worth of traffic Measured in fixed-size chunks
T_{tw}	Target Time Window (set to 10 secs here)
TSI, c	Time Since Inception (Header field of *Request*): Hop-Distance from Client, Value range: $1\ to\ n$
TSB, x	Time Since Birth (Header field of *Content*): Hop-Distance from Server, Value range: $1\ to\ n$
N_x	Cache size of router x Router x is TSB hops away from content source

memory to cache contents in a (RL)DRAM chip for at least one second (see third column in Table 1).

- Authors in [6] show up to 60 percent bandwidth savings by redundant traffic elimination *within the first 10 seconds after the original transmission*, in some enterprise networks. We associate redundant traffic, i.e., subsequent requests for the same content, with the aforementioned figure. That is, we consider, without loss of generality, that any content should be kept in *any* one of the path's caches for a *target time window*, T_{tw}, of 10 seconds.

Both the above settings are relatively arbitrary and can change in the future, but these values are a good starting point based on today's technology. We note that our concepts and algorithms presented next are still applicable should these values change. For example, the setting for the target time window is used here to obtain benchmark results. Larger values for T_{tw} will result in contents staying in the cache for longer, while smaller values will result in more evictions. We briefly evaluate different values for T_{tw} in Section 5.2, but we note that the setting for the target time-window depends also on the traffic characteristics of the domain (e.g., temporal locality characteristics of the traffic [25]) and therefore, it is up to the ISP to set this value for its own network.

2.2 System Model

We assume the topology of Fig. 1 and consider (for simplicity) that all network routers also have caching capabilities. Note that the proposed scheme does not necessarily require all routers to be caches. The decision (of an ISP) as to which nodes to turn into cache routers depends on several parameters, such as the topology, the number of inter-domain links and the traffic characteristics. Our recent study that estimates the proportion of time that a given content stays in-cache given its relative request rate [18] can assist on this direction.

In Fig. 1 the path from source to destination comprises n routers, where router r_i has N_i cache slots, each able to hold one addressable content chunk (one-to-one relation of chunks and cache slots). We assume that content chunks are of fixed size, similarly to the fixed size of an IP packet or the MTU; based on the discussion above, we assume that N_i slots can hold one second worth of traffic. Our model notation is given in Table 2. We also assume a *Request-Response* model of Information-Centric Networks, where *Request* and *Content* messages follow the same route (symmetric routing), similarly to recent proposals, such as [1], [2], [21], [26]. This is a fair assumption in general, given the name-based, *location-independent* routing assumed in Information-Centric Networks. We introduce the following concepts:

Path Cache Capacity. The *caching capacity* of the path of caches is $\sum_{i=1}^{n} N_i$ in terms of memory, which amounts to n seconds worth of traffic cached along the delivery path.

Path Cache Capability. Given that our target time window is T_{tw} seconds worth of traffic cached along a given path, the *caching capability* of an n-hop long path, as a fraction of the required capacity for T_{tw} seconds, is $\frac{\sum_{i=1}^{n} N_i}{T_{tw} N}$, where N is the average cache size along the path. We revisit the issue of average cache size in the next section.

Path Length Monitoring. Similar to the Time To Live (TTL) field included in IP packets, our design requires that ICN *request message* headers include the *Time Since Inception* (TSI) field and *content message* headers include both the TSI and the *Time Since Birth* (TSB) fields. Every router increases the TSI value of request packets by one. The content source attaches the TSI value that it sees on the request message to the content message. Every router increases the TSB value of the content message by one, as shown in Fig. 1. Hence, during the content message's journey back to the client, the TSI value in the content message is a fixed value and denotes the path-length of this specific content flow, while the TSB value denotes the number of hops that the content message has travelled so far. In case of a cache hit, the TSI and TSB values are treated as if the cache is the actual source, that is, the TSI value of the content message is replaced by that of the *Request* message, while the TSB is set to 1. Furthermore, in case of request aggregation, TSI starts counting from the aggregation point, that is, TSI counts as if the requesting client was attached to the aggregation node. The rationale behind this decision is that concurrent requests for a content reveal high content popularity, hence, it is better to cache this content at central nodes, such as the aggregation node or its neighbors.

3 BUILDING *PROBCACHE*

We approach the problem of content placement within a system of caches from the *path caching capability* point of view. In particular, *each router, based on the amount of traffic that it has to serve per unit time, indirectly approximates the number of copies of incoming contents that the (rest of the) path can accommodate*. This value is the $TimesIn$ factor (see Section 3.1). Based on the $TimesIn$ indication and on *the router's distance from the user*, which we call $CacheWeight$

(see Section 3.2), each router probabilistically caches contents as they travel along the path (see Section 3.3). $ProbCache$ was initially introduced in [19]. In this study we elaborate on the behavior of the algorithm's components and we propose $ProbCache+$ (in Section 4.2), an enhancement to the original algorithm that allocates cache resources to flows in a fairer manner. We note however, that the main concepts introduced in this section remain the same for both the original version of $ProbCache$ [19] and for its enhanced version.

3.1 Estimating the Caching Capability of a Path

Consider two users shown, in Fig. 1, five and four hops away from the server, respectively. The total cache capacity of the path is $\sum_{i=1}^{n} N_i$, where $n_1 = 5$ for $Request_1$ and $n_2 = 4$ for $Request_2$. Grey circles denote the caches that have to be shared between the two users, while white and black circles denote caches used exclusively by Users 1 and 2, respectively.

The number of times that the path can afford to cache this content chunk is reflected in the $TimesIn$ factor, whose calculation takes place as follows:

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

where c is the *Time Since Inception* (TSI) value and x is the *Time Since Birth* (TSB) value that the router sees in the header of the content message (Table 2). Therefore, in Eq. (1), N_x denotes the size of cache x, TSB hops away from the source. As an example, consider content messages traveling through router r_2 to fulfil $Request_1$, in Fig. 1; these messages will have $TSI = 5$ and $TSB = 2$, while contents for $Request_2$ will have $TSI = 4$ and $TSB = 2$. Given that the sum in Eq. (1) is calculated in every node the content traverses, it considers the result of the subtraction of TSI minus TSB [or $c - (x - 1)$ in Eq. (1)], to account for the *remaining* caches only, instead of the total number of caches from the content source to the client. In Fig. 1 for example, and for a content chunk in r_3 that travels to User 1, the $TimesIn$ value refers to the white circles, in order to leave the grey circles for users connected closer to the source.

3.2 Weight-Based Caching

We argue that in order to achieve fair resource (in our case cache) allocation in a distributed environment, each content flow has to take into account other content flows sharing the same path (grey circles in Fig. 1). Hence, to decide *where* to cache the number of copies that $TimesIn$ indicated, we use the *Cache Weight* factor

$$Cache\ Weight(x) = \frac{x}{c} \tag{2}$$

where x is the TSB value of the packet header and c is the TSI value; therefore, $Cache\ Weight \in [\frac{1}{c}, 1]$. We note that the TSI value is fixed during the content chunk's journey from the source to the client, while the TSB value is increasing for each router the chunk traverses; hence, $CacheWeight \to 1$ as the content chunk is getting closer to its destination. This is a desirable system property considering path-diversity, in terms of hops, between different client-source pairs.

3.3 ProbCache: Probabilistic In-Network Caching

ProbCache is the product of $TimesIn$ and $CacheWeight$ as shown in Eq. (3). Each router along the path caches incoming chunks according to $ProbCache(x)$, denoted as $P(x)$ in all equations, depending on their TSI and TSB values

$$P(x) = \underbrace{\frac{\sum_{i=1}^{c-(x-1)} N_i}{T_{tw} N_x}}_{TimesIn} \times \underbrace{\frac{x}{c}}_{CacheWeight}. \tag{3}$$

The intention behind the *Cache Weight* factor is to increase the probability of a content being cached closer to its destination. This way, we expect to achieve *fair content flow multiplexing* between contents that travel to different destinations in terms of path length. For example, contents for User 1 in Fig. 1 should be cached inversely proportionally to User 1's distance from the server, i.e., in (white) routers r_4, or r_5, in order to leave (grey) routers $r_1 - r_3$ for clients travelling shorter paths to cache their contents. This is in accordance to our previous findings in [18] that contents tend to be cached for longer towards the edge of the network.

In Section 4, we evaluate the effect of the $CacheWeight$ factor in combination with $TimesIn$. We find that in contrast to our expectation, $ProbCache$ behaves unfairly despite the effect of $CacheWeight$. We proceed to fix the unfair behavior of $ProbCache$ in Section 4.2 by modifying $CacheWeight$ and proposing an enhanced version of the algorithm, which we call $ProbCache+$.

We note the following regarding the overall design of $ProbCache$ (which also apply to $ProbCache+$ introduced later in this paper):

1. The $TimesIn$ factor may indicate that more than one copies of a content chunk can be accommodated along the content delivery path. This is especially so in case of long-haul transmissions, where content flows travel through many hops before they reach their destination. Therefore, $ProbCache$ is not bounded to a maximum value of 1, but can take on bigger values depending on: 1) the path-length, and 2) the target time window (T_{tw}—see Table 2 and related discussion in Section 2.1). The theoretical upper bound of $ProbCache$ is the number of hops from source to destination for each content flow, that is, the extreme and rather unrealistic case where there is enough cache capacity along the path to keep one copy at each node the chunk is going through. In this extreme case, the operation of $ProbCache$ is similar to that of universal caching.
2. To calculate the $TimesIn$ factor each router has to conjecture on the size of the rest of the caches on the path. However, given that we do not know what amount of cache each router will have, or if backbone routers, for instance, will have bigger caches than edge-network routers, we make the following simplifying assumption. *Each router assumes that all other routers on the path have the same amount of cache as it has got*. Even in a random-size cache deployment scenario, this assumption serves our purposes well. That is, a router with a big cache, compared to the caches along the path, will be caching contents with higher

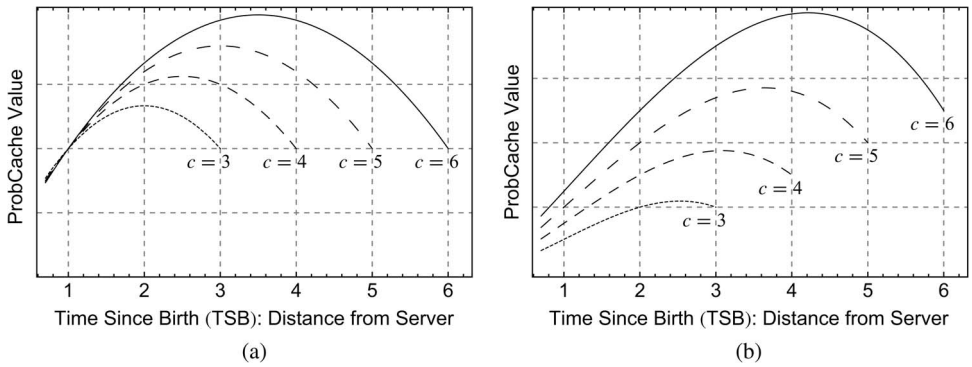

Fig. 2. Behaviour of $P_{c\rightarrow e}(x)$ and $P_{c\rightarrow E}(x)$ along a six-hop delivery path. We observe that the longer a flow's distance is from the content source, the higher the chances it has got to cache its contents along the whole path. This is in contrast to our design considerations for fair content multiplexing. (a) Behaviour of $P_{c\rightarrow e}(x)$, Eq. (3). (b) Behaviour of $P_{c\rightarrow E}(x)$, Eq. (9).

probability, while a router with a small cache will experience the opposite effect [Eq. (1)]. This is a desirable system property which alleviates the effect of unknown cache sizes, but at the same time guarantees fair load distribution among nodes with diverse amounts of cache memory. In [19], we have shown that although our simplifying assumption does not harm the performance of *ProbCache* in heterogeneous cache size environments, it fails to fully exploit extra caching resources.

3. $ProbCache$ is a raw number that does not have a unit. The raw number essentially represents the caching probability of incoming content chunks. Although T_{tw} is physically interpreted in terms of time (and therefore, the formula in Eq. (3) points to $\frac{1}{second}$ for the unit of the function), it is essentially a raw number, which acts as a weight factor to determine the replacement granularity of contents in the system of caches. The smaller the value is the faster items are replaced in caches, as discussed earlier in Section 2.1.

In [19], we have modified $ProbCache$ for the case of heterogeneous cache sizes along the delivery path. In particular, we have considered three cache size settings: homogeneous cache sizes, larger caches towards the core and larger caches towards the edge of the network [27]. We provide a summary of our findings in Supplementary Section 8, where we also present the modification of $ProbCache$ for heterogeneous settings. We refer the reader to that section for a more detailed presentation of these versions of $ProbCache$. In the rest of the paper, we focus on two cache size settings, namely homogeneous caches and larger caches towards the edge and evaluate the behavior of $ProbCache$ based on these.

4 Theoretical Assessment

4.1 Analysis of Basic Functions

Our intention is to explore the behavior of $ProbCache$ for homogeneous cache sizes [$P_{c\rightarrow e}(x)$, Eqs. (3) and (7)] and heterogeneous cache sizes [$P_{c\rightarrow E}(x)$, Eq. (9)] with regard to its resource management and utilization properties. We are interested in *fair and efficient* content flow multiplexing in in-network caching architectures, which would in turn increase the overall network performance.

We begin by plotting the behavior of the two versions of $ProbCache$ for users connected at different points along a sample six-hop path in Fig. 2.

We observe that although fair content multiplexing was one of the main design considerations for $ProbCache$ [19], and the main operational property of $CacheWeight$, in reality, the path capacity calculation ($TimesIn$ factor) has unexpectedly high impact on the overall behaviour of the algorithm. That is, we see that the longer the distance from a client to the source of content, the higher the probability this client has to cache contents along *the entire path* (i.e., even far from its attachment point and closer to the source; nodes 1 to 4 in Fig. 2). Clearly, this behaviour leads to unfairness in terms of resource allocation between clients. In Fig. 3 we also plot the behaviour of $ProbCache$ for variable T_{tw} (from 5 to 20) and with constant $c = 6$, where we see that although the absolute values of the function vary with different T_{tw} the overall trend is similar to the default value of 10.

To verify this observation and get a deeper understanding of the operational properties of the algorithm, we elaborate on the distribution of values that the function gets along a path of caches for clients connected at different points along the path. We initially calculate the derivatives of the two versions of the algorithm, in Supplementary Section 9.1 to see at which point along the path $ProbCache$ gets its maximum values; we also calculate the integrals of $ProbCache$ in Supplementary Section 9.2 to monitor the density distribution of the value range of $ProbCache$ along the delivery path. Our analysis in Supplementary Section 9.2 validates our observation in Fig. 2 that $ProbCache$ behaves unfairly for flows connected at different points along the delivery path and gives more caching opportunities to flows connected far from the content source.

Based on those findings, we proceed to investigate further and fix the behaviour of $ProbCache$ in the next section; we base our analysis on the versions of the algorithm that apply to homogeneous caches and for larger caches towards the edge of the network.

4.2 Enhancing the Content Multiplexing Fairness of ProbCache

Fig. 2 shows that $ProbCache$ increases the probability of contents being cached very early compared to the distance of the client to the source. In contrast, according to our

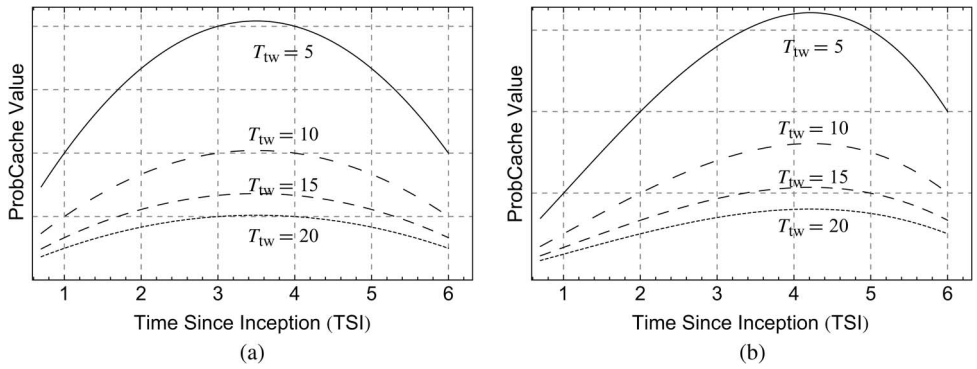

Fig. 3. Behaviour of $P_{c\to e}(x)$ and $P_{c\to E}(x)$ along a six-hop delivery path with variable T_{tw} for $c = 6$. (a) Behaviour of $P_{c\to e}(x)$ with variable T_{tw}, Eqs. (3) and (7). (b) Behaviour of $P_{c\to E}(x)$ with variable T_{tw}, Eq. (9).

design principles, the ideal algorithm would increase its value directly proportionally to the distance of the client from the source. That is, it would get low values when going through the first nodes of the path and would increase as it would get closer to its destination. To accommodate this requirement and improve the content multiplexing fairness of $ProbCache$ we apply the following modification: we raise the value of $CacheWeight$, $\frac{x}{c}$), to the power of c (i.e., the TSI value of the delivery path), $(\frac{x}{c})^c$.

Our modified $ProbCache$ functions, which we denote as $P'_{c\to e}(x)$ and $P'_{c\to E}(x)$ for the homogeneous and the heterogeneous case, respectively are therefore, the following:

$$P'_{c\to e}(x) = \frac{N_i(c-x+1)}{T_{tw}N_x}\left(\frac{x}{c}\right)^c = K_1(c-x+1)\left(\frac{x}{c}\right)^c \quad (4)$$

where $K_1 = \frac{N_i}{T_{tw}N_x}$, and

$$P'_{c\to E}(x) = \frac{1}{2}\frac{N_i}{T_{tw}N_x}(c^2-c-x^2+3x-2)\left(\frac{x}{c}\right)^c \Rightarrow$$
$$P'_{c\to E}(x) = K_2(c^2-c-x^2+3x-2)\left(\frac{x}{c}\right)^c \quad (5)$$

where $K_2 = \frac{1}{2}\frac{N_i}{T_{tw}N_x}$.

We plot the behaviour of these enhanced versions of $ProbCache$ in Fig. 4. Clearly, the performance of the algorithms is different now and much closer to what we expected. Depending on the attachment point of the clients, the value of the algorithm adjusts accordingly, in order to leave resources closer to the source of the content for clients travelling shorter paths. In Fig. 5, we plot the behaviour of the enhanced version of $ProbCache$ for variable values of T_{tw} and constant $c = 6$. We see, similarly to Fig. 3, that the function behaves as desired for different values of T_{tw}, although the absolute value of the function varies, as also discussed in Section 2.1. In Supplementary Section 10 of the Supplementary file, we repeat the methodology followed before to verify our initial observations. We initially calculate the derivatives of both versions of the enhanced algorithm; we equal the derivatives to zero to find the maximum values of $P'_{c\to e}(x)$ and $P'_{c\to E}(x)$, which we plot in Fig. 12; finally, we calculate and plot the integrals to evaluate the density distribution of the enhanced version of $ProbCache$.

4.3 Summary of Theoretical Analysis and Findings

We have elaborated on the performance of the original version of $ProbCache$ [19] and found that it favors content flows that connect far from the content source, over content flows that travel shorter distances (Fig. 2). We verified this observation by capturing the value density distribution of the basic equations of the algorithm, where we saw that longer content flows (in terms of hops from source) behave more aggressively and therefore, attempt to occupy resources along the entire delivery path. Based on these findings, we have modified the formula of $ProbCache$ in order for its value density distribution to evolve according to path lengths (Fig. 4). Our theoretical findings show that the new version of $ProbCache$ [which we denote from now on as $ProbCache+$ and is given in Eqs. (4) and (5)] is indeed

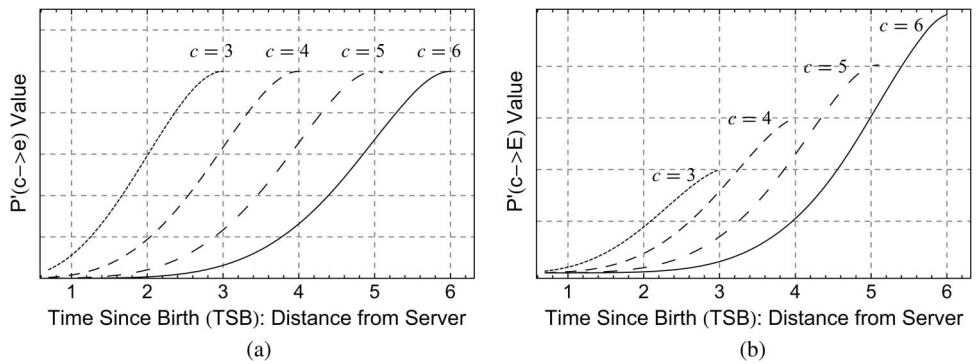

Fig. 4. Behaviour of $P'_{c\to e}(x)$ and $P'_{c\to E}(x)$ along a six-hop delivery path. The modified $CacheWeight$ factor [Eqs. (4) and (5)] results in caching probability directly proportional to the node's distance from the source. This is in accordance to our initial design principles. (a) Behaviour of $P'_{c\to e}(x)$, Eq. (4). (b) Behaviour of $P'_{c\to E}(x)$, Eq. (5).

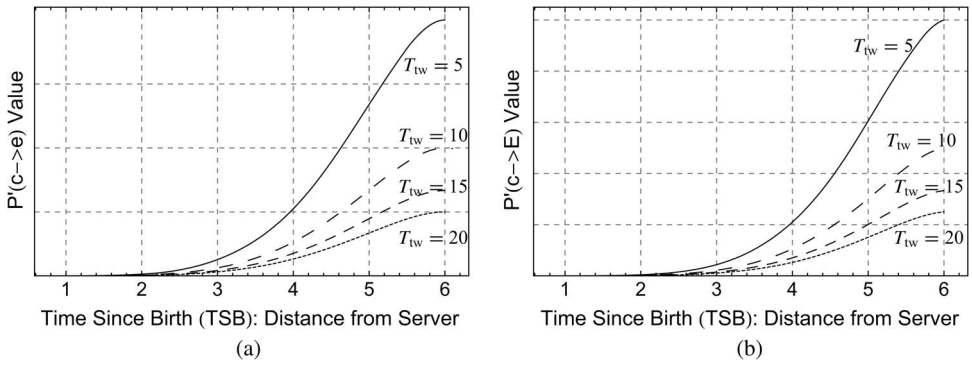

Fig. 5. Behaviour of $P'_{c\to e}(x)$ and $P'_{c\to E}(x)$ along a six-hop delivery path with variable T_{tw} and $c = 6$. (a) Behaviour of $P'_{c\to e}(x)$, Eq. (4). (b) Behaviour of $P'_{c\to E}(x)$, Eq. (5).

more fair in terms of resource allocation and content flow multiplexing along a path of in-network caches.

5 Performance Evaluation

5.1 Simulation Environment and Setup

We test our algorithm in a custom-built simulator, where we use *Least Recently Used* (LRU) caches. Given that the ultimate goal of *ProbCache* is to manage caching resources more efficiently, by reducing caching redundancy, the straightforward metric of interest is the *reduction of Server Hits*. Furthermore, the gain from serving user requests from intermediate caches, instead of travelling to the origin server, depends on the number of hops that the request travels before it eventually hits cached contents. Clearly, as the number of hops increases, the overall gain decreases. To measure this gain, we also monitor the *Hop Reduction Ratio*. Our simulator, which we also make publicly available in [28], is an event-based simulator, hence, all metrics are explicitly measured at the end of the simulation. Cache Hits achieved by all nodes are summed up, while the Hop Reduction Ratio is summing the distance (in terms of hops) to the nodes where hits have happened over the sum of the distance to the origin server.

Apart from the above well-known metrics, which have been widely used in caching research in the past, we also introduce one extra metric in order to capture the *resource management and content flow multiplexing properties* of in-network caching algorithms. We call this metric *Content Multiplexing Fairness Index* (CMFI) and we define it as *the amount of caching resources that the algorithm under consideration has left unused in order for other content flows to exploit over the total amount of cache resources available along the entire delivery path*. The formula according to which the *CMFI* index is calculated is given below in Eq. (6) for the homogeneous and heterogeneous cache deployment cases, respectively

$$CMFI_{c\to e} = \frac{\sum_1^x N_i}{\sum_1^c N_i} = \frac{xN_i}{cN_i} = \frac{x}{c}$$
$$CMFI_{c\to E} = \frac{\sum_1^x iN_i}{\sum_1^c iN_i} = \frac{x(x+1)}{c(c+1)}. \quad (6)$$

The rationale behind the design of $CMFI$ is as follows: considering that a relatively realistic representation of the Internet topology comprises of fewer nodes at the core, which then fan out to more leaf/edge routers that finally reach out to residential connections, the *CMFI* index considers fairer to cache towards the edge of the network, rather than in the busy core. Therefore, the index gets bigger values for algorithms that tend to cache towards the edges of the network.

Although someone may argue that by caching contents at the edge of a delivery path implicitly restricts access to those contents by clients connected further up towards the source, we consider that increased demand for content from across the delivery path will minimize the effect of this argument.

Furthermore, we highlight the following important property of the index introduced here: *CMFI* is not a flow-oriented metric, but rather a content-oriented one. It targets multiplexing of *contents* (and not of flows) in caches. That said, an algorithm that does not cache any content in any cache is 100 percent fair (when compared to itself), as it leaves space for other contents to be cached-this is similar to a transport protocol that forces all flows to consume a tiny portion of the available bandwidth, which is fair, but still rather inefficient. The related flow whose contents are not cached is not treated unfairly, as contents will be delivered to this flow anyway. Unfairness here comes in terms of the content, rather than the respective flow i.e., the content will not be available in the cache network for potential future requests.

We use scale-free topologies following the Barabasi-Albert model [29], where nodes follow power-law degree distribution to reflect realistic Internet topologies. We use a benchmark topology of 200 nodes, but we note that results remain qualitatively similar in larger topologies of up to 700 nodes.

We set the exponent of the Zipf distribution of content popularity (a) to 1.2 to capture the case of medium-popularity content [30], while requests are generated following Poisson distribution. Again, content popularity affects the results in a quantitative way, as can also be verified from the results presented in our previous study in [19], where we have set Zipf $a = 0.8$. That is, although higher popularity (i.e., exponent of Zipf distribution) results in higher performance for caching algorithms, this improvement is similar for all algorithms tested. Hence, qualitatively, the results follow similar trends for exponents between 0.5-1.5. As we show later in this section,

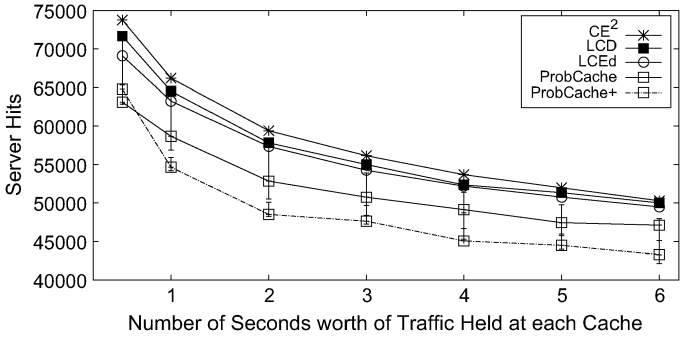

Fig. 6. Scenario 1: Server hit performance, set of simulations applying increasing amount of cache in each experiment.

when popularity exceeds a certain threshold and causes small-scale *flash-crowd* events, then performance is affected more drastically. We discuss such issues in Section 5.3.

The experiments run until 100,000 content requests have been successfully completed. We use two content servers, which are connected at two different nodes in the core of the network, that is, at the most well-connected part of our topology. This placement of servers was done on purpose in order to: 1) reflect a relatively realistic network topology,[1] and 2) avoid overwhelming isolated nodes (by attaching busy servers next to them), as this would (negatively) impact the behaviour of some of the protocols, as discussed further later on. We set the *cache-to-catalogue size* ratio to 0,0001-0,001 percent and as discussed before, we set the default cache size to be one second worth of traffic transmitted through the incoming link at each router.

We compare the following caching strategies:

1. universal, or ubiquitous caching, a scheme that we call *Cache Everything Everywhere* (CE^2) (and was implicitly supported in [2]);
2. the *Leave Copy Down* (LCD) [20] algorithm proposed in the past for overlay caching topologies. According to LCD, cache hits cause contents to be copied one hop closer to the user, or one level down the cache hierarchy;
3. the *Leave Copy Edge* (LCEd) algorithm. LCEd caches contents deterministically one-hop before the client. According to the main design principle of $ProbCache$, contents should be cached closer to their destination with higher probability, in order to leave caching space at the core of the network for shorter content flows. Therefore, one might contend that a simpler algorithm that caches contents at the very edge of every content-flow path might achieve similar results. We, therefore, include LCEd in our performance evaluation for completeness;
4. the original version of $ProbCache$, Eq. (3), [19];
5. the enhanced version of $ProbCache$ [Eqs. (4) and (5)], which we introduced in the present study.

1. Although content sources and big content providers are not necessarily placed in tier-1 domains, most CDNs are placed towards the core of the network from the view-point of the clients.

We note that in [19] we have evaluated the performance of $ProbCache$ against simpler approaches to in-network caching, e.g., with probabilistic algorithms that cache with probability $p = 0.7$ and $p = 0.3$ at every cache. Our evaluation showed that such algorithms perform similarly to LCD, hence, we omit further comparison against such approaches in the present study. Modulo caching [31], an algorithm proposed in the past in the area of active networks, caches contents along the path according to a *modulo* calculation. This results in caching content messages every few hops along the delivery path. Although this approach might seem close to our design, we highlight that it follows a resource-management-agnostic approach and therefore, performs similarly to LCD or the fixed probability algorithms evaluated in [19]. For this reason, we have chosen not to include *modlulo-caching* [31] in our evaluations in this paper.

We present three evaluation scenaria, which capture the most important aspects of the behaviour of the above-mentioned caching protocols. In the first scenario (Section 5.2), we test the performance of the algorithm in an homogeneous cache-size environment, where all nodes in the network cache traffic for one second. In the second scenario (Section 5.3), we assess the performance of the caching protocols with regard to their convergence properties to popular content, that is, how fast are the protocols getting aware of a content that is becoming popular in order to keep copies of it in several caches. Finally, in our last evaluation scenario, which is presented in Supplementary Section 11 we use heterogeneous caches along the paths and in particular, we assume larger caches towards the edges of the network.

5.2 Scenario 1: Homogeneous Cache Environment

We evaluate the performance of the five caching strategies in a homogeneous cache-size deployment in the 200-node topology. In Fig. 6, we present the Server Hits performance of the five protocols. We see that $ProbCache$ outperforms the rest of the protocols by approximately 4-5 percent, while $ProbCache+$ reduces the number of Server Hits by an additional 5.5 percent compared to the original $ProbCache$ algorithm. This means an overall performance difference of approximately 10-11 percent for $ProbCache+$ compared to CE^2, LCD and LCEd, which perform largely the same.

In the same Figure we also evaluate the performance of both $ProbCache$ and $ProbCache+$ for different values of $T_{tw} = [1, 5, 10, 20]$ (default value is 10). The results are presented in terms of errorbars in the lineplots of $ProbCache$ and $ProbCache+$. In particular, the errorbars above the lineplots depict the performance of the protocols when $T_{tw} = 1$, while the errorbars below the lineplots show the performance when $T_{tw} = 5$. For $T_{tw} = 20$ the performance is identical to the default setting. These results reveal the following based on the specific settings of this experiment: 1) Small values of T_{tw} (equal to 1 in our case) result in many cache evictions and degrade the overall performance. 2) Large values of T_{tw} (equal to 20 here) do not affect the performance, as the caching probability is very similar to the default setting. 3) $T_{tw} = 5$ seems to achieve better performance than the default setting, especially for the original version of $ProbCache$ [19]. 4) $ProbCache+$ seems to

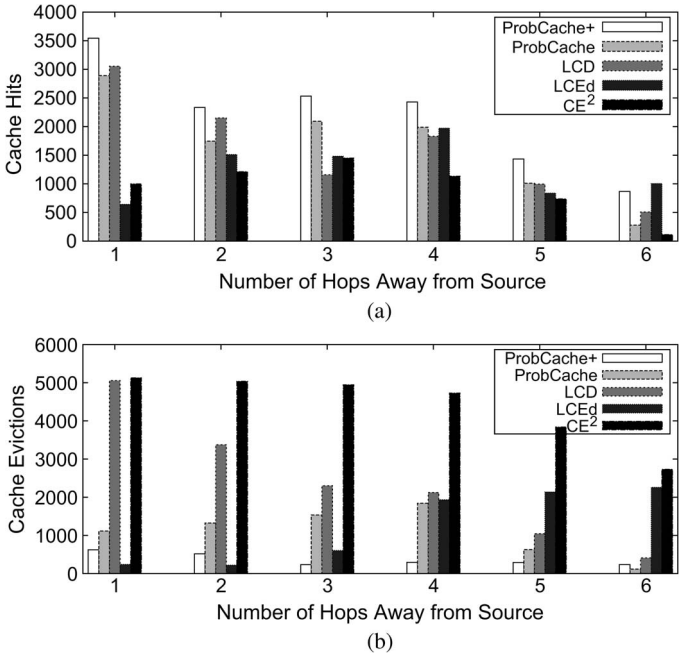

Fig. 7. Scenario 1: Cache hits and cache evictions along one delivery path. (a) Per hop cache hits. (b) Per hop cache evictions.

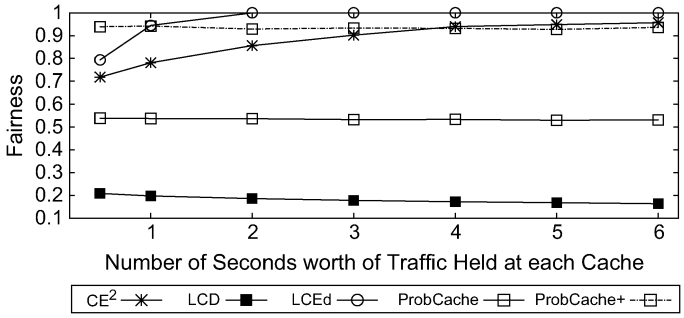

Fig. 8. Scenario 1: Content Multiplexing Fairness Index [CMFI, see Eq. (6)].

be less affected by the setting of T_{tw} than $ProbCache$. Although interesting, we stress that these observations regarding different values of T_{tw} should not be considered as conclusive statements for every network setup. Further investigation and experimentation is needed before general conclusions can be drawn.

To get a better understanding of the performance of the protocols, we trace the per-hop *Cache Hits* and *Cache Evictions* of the protocols along a random 6-hop path in our topology. The results are presented in Fig. 7. Figure 7a depicts the overall cache hit behaviour of the protocols, while Fig. 7b focuses on cache evictions. For example, we see in Fig. 7b that LCD, in line with its core design principles, is pretty aggressive in evicting contents from caches close to the source of the content. In contrast, as we move away from the source LCD is becoming less aggressive. This behaviour results in more hits (and more evictions) at the core of the network (i.e., close to the server) and less towards the edges. These hits, however, are not enough in order to guarantee satisfactory overall performance as shown in Fig. 6. On the contrary, LCEd caches always at the edge of the path and hence, results in more aggressive behaviour (and therefore, more evictions) at the edges of the network. Both algorithms, however, always cache at least one copy of the content along the path an action that might not always be affordable, given the limited caching resources available. Furthermore, LCD and LCEd cache at fixed and predefined places along the path, resulting in poor resource management and content distribution in the pool of available caching space.

The sophisticated resource management approach of $ProbCache$ and $ProbCache+$ clearly makes better use of available resources as shown in Figs. 7a and 7b. We see a clear difference between the cache evictions of the two versions of $ProbCache$ and the rest of the protocols (Fig. 7b). The resource management framework proposed here and integrated in both versions of the protocol caches incoming content chunks according to the resource availability along the path. This means that not all content chunks that traverse a path get necessarily cached in one of the router-caches, but instead, some chunks might not get cached at all.

Elaborating on the performance of the two versions of $ProbCache$, we observe the following. The original version of $ProbCache$ [19], due to its slightly unfair content flow multiplexing behaviour (also shown in Figs. 2a and 2b) tends to cache contents half-way through the path from source to destination. This is clear from Fig. 7b, where nodes 2, 3, and 4 are forced to evict the highest number of chunks. Although this does not necessarily translate to less cache hits, as shown in Fig. 7a for nodes 2, 3, and 4, it does impact the overall performance of the protocol (see Fig. 6). In contrast, the enhanced version of the algorithm (see (4) and (5) and Fig. 4 in Section 4.2), which targets more fair resource allocation along the path achieves less evictions in all nodes along the path, as shown in Fig. 7b. This verifies our claims in the previous Section and in Supplementary Section 10, where we presented the density distribution of the algorithm (see Fig. 13 of the Supplementary sections), as well as the *local maxima* distribution (see Fig. 12 of the Supplementary sections), where both plots fall within the *fair content distribution area* (grey area in Figs. 12 and 13 of the Supplementary sections).

As a final step to verify our theoretical findings regarding the multiplexing behaviour of the two versions of $ProbCache$, we plot the *Content Multiplexing Fairness Index* (CMFI) introduced earlier. The result is shown in Fig. 8.

We see that the design principle of LCD to gradually move contents down the cache-path towards the edge of the network results in low content multiplexing capability. In contrast, LCEd, by caching contents only at the edges of the network, leaves caching space for other content-flows. However, the deterministic nature of LCEd results in lower overall performance (e.g., in terms of server hits) as we have shown in Fig. 6. Finally, the original version of $ProbCache$ shows little potential for fair content multiplexing, in contrast to our original design goals and in accordance to our findings in Section 4. $ProbCache+$ on the other hand, exhibits ideal content multiplexing performance, by caching contents according to their path lengths, but also according to the path cache capability. These

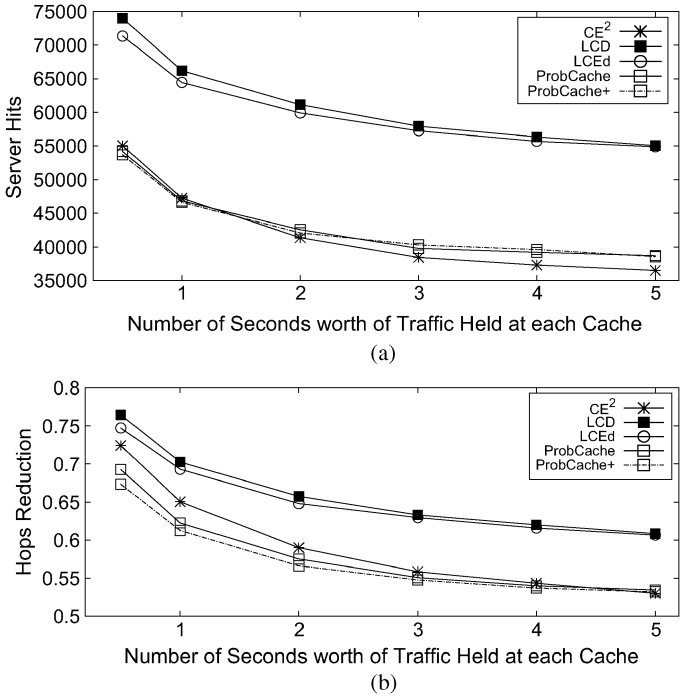

Fig. 9. Scenario 2: Caching convergence to popular content-Set of simulations applying increasing amount of cache in each experiment. (a) Server hits. (b) Hop reduction ratio.

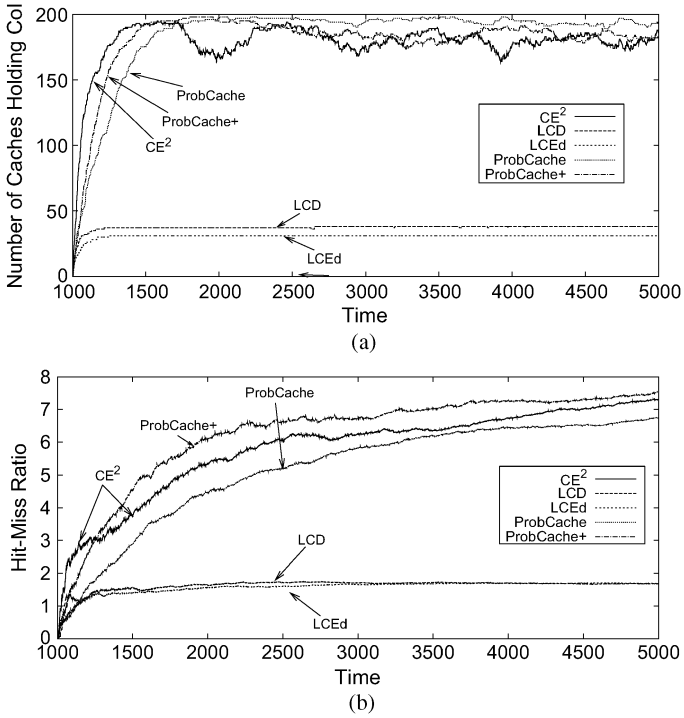

Fig. 10. Scenario 2: Caching convergence to popular content. (a) Number of caches holding *CoI*. (b) *Hit-miss* ratio.

results verify the theoretical analysis of the two versions of $ProbCache$ in the previous Section.

5.3 Scenario 2: Caching Convergence to Popular Content

In this scenario, we test the performance of the caching protocols with regard to their convergence time in case of emerging popular content. That is, we consider that after the 1000th second of the simulation, a previously unpopular content, which we refer to as the *Content of Interest* or *CoI*, becomes popular and constantly receives 5 percent of the total requests generated per second. We use the same 200-node topology and the homogeneous cache size setup to capture the behaviour of the algorithms in the basic setting. We note that the proposed algorithms are inherently popularity-agnostic. The results are summarized in Figs. 9 and 10.

In Fig. 9, we see significant performance differences between $ProbCache$, $ProbCache+$, CE^2 (which perform roughly the same) and LCD, LCEd, both in terms of Server Hit ratio (which now is very close to 20 percent, Fig. 9a) and in terms of Hop Reduction Ratio (which is approximately 10 percent, Fig. 9b). In particular, LCD and LCEd being deterministic algorithms that cache one copy of every content in specific points along the content delivery path fail to exploit the popularity of the *CoI*. This owes to the fact that the deterministic nature of caching in these cases results in contents (and particularly the *CoI*) being evicted from the cache before it receives further requests. CE^2, which also caches deterministically, escapes (to a certain extend) this behaviour due to its inherent caching redundancy feature.

Although someone might expect that caching contents probabilistically might result in slow convergence and therefore, reduced performance, both versions of $ProbCache$ invalidate this claim. The sophisticated design of $ProbCache$ (and $ProbCache+$) bases its probabilistic caching behaviour on the amount of traffic served by the router *per unit time* and not based on arbitrary assumptions regarding the server catalogue size. Having said that, $ProbCache$ is essentially choosing contents from a small range of incoming traffic (see Eqs. 7, 8, and 9 in Section 9). This feature of the proposed algorithm reduces the caching redundancy and increases the number of contents cached along the entire path. Therefore, although popular content might not get immediately cached in all routers along the path, it still gets cached at one point along the delivery path and therefore avoids being fetched from the origin server.

To prove our claims further, we plot *the number of caches that hold the CoI* in Fig. 10a and the *hit-miss* ratio in Fig. 10b as the experiment progresses. In Fig. 10a, we observe that indeed CE^2 populates the caches faster with the *CoI* compared to $ProbCache$ and $ProbCache+$, but this difference is in the order of 120 to 150 seconds, as can be seen in the beginning of the experiment from the 1050th to the 1200th second. In the long term, however, we see that CE^2 evicts the *CoI* faster than new requests come in and therefore results in less cache hits. This is shown in Fig. 10b, where although for the interval between the 1050th and the 1200th second CE^2 has higher *hit-miss* ratio, after the 1200th $ProbCache+$ is performing better. With regard to LCD and LCEd, we see in Fig. 10 that the deterministic approach to content caching results in very poor performance in terms of convergence to caching of popular content. This behaviour is due to high contention for caching slots in the fixed places where these algorithms cache contents. In turn, this results in evicting the *CoI* faster than new requests come in.

6 Summary and Conclusion

We have argued that caching named chunks in network routers' DRAM memory, as opposed to caching large objects or files in proxy disks, calls for reconsideration of past approaches to caching. In-network caching in ICNs has to happen in an uncoordinated and distributed fashion, taking also into account the available cache resources. We have therefore, introduced the concept of *resource management* for in-network caching environments and have argued that indiscriminate and/or deterministic caching presents little potential for efficient resource utilization.

We have proposed *ProbCache*, an algorithm that approximates the capability of paths to cache contents, based on path lengths, and multiplexes content flows accordingly. The ultimate goal of *ProbCache* is to utilize resources efficiently, reduce *caching redundancy* and in turn, *network traffic redundancy*. We have considered both homogeneous and heterogeneous cache sizes and have adjusted *ProbCache* to fit in both environments.

Although the calculation of the value of *ProbCache* introduces extra computation complexity, we report savings of up to 20 percent in server hits; 7-8 percent in the number of hops to hit cached contents; and reduction by an order of magnitude in cache evictions. We argue that the reduction of cache evictions counter-balances the extra computation cost of *ProbCache*.

Several issues in our design warranty further investigation. For example, our formula can be adjusted to enforce caching towards the edge of a domain and before an expensive transit link. Furthermore, in this study, we have not considered selective caching in terms of content importance. Given the long tail distribution of Internet traffic, as well as temporal locality characteristics, we consider that only a subset of contents are worth caching inside the network. Content identification policies can be based on expected content popularity, which can be driven by SLA agreements between content providers, CDNs and ISPs. We hope that our study will trigger further research as the ICN research field matures.

Finally, security and billing issues constitute major open issues in the area of ICN. Clearly, by scattering contents along the content delivery path, *ProbCache* avoids both *cache polution* and *sniffing* attacks [32]. This is because the attackers are not able to locate the exact cache where sequences of chunks of the same content might be cached. As regards billing issues between ISPs and CDNs, we believe that the *target time window* value (T_{tw}) of *ProbCache*, together with our previous modeling work on the proportion of time that a given content stays in the network's caches [18] can assist in the design of a billing framework for ICN. We leave those topics as directions for future work.

References

[1] T. Koponen, M. Chawla, B.G. Chun, A. Ermolinskiy, K.H. Kim, S. Shenker, and I. Stoica, "A Data-Oriented (and Beyond) Network Architecture," in *Proc. SIGCOMM*, 2007, pp. 181-192.

[2] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, and R.L. Braynard, "Networking Named Content," in *Proc. ACM CoNEXT*, 2009, pp. 1-12.

[3] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker, "Naming in Content-Oriented Architectures," in *Proc. ACM SIGCOMM ICN Workshop*, 2011, pp. 1-6.

[4] A. Detti, M. Pomposini, N. Blefari-Melazzi, and S. Salsano, "Supporting the Web with an Information Centric Network that Routes by Name," *Comput. Netw., pp. 3705-3722, Aug. 2012.*

[5] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker, "Packet Caches on Routers: The Implications of Universal Redundant Traffic Elimination," in *Proc. ACM SIGCOMM*, 2008, pp. 219-230.

[6] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in Network Traffic: Findings and Implications," in *Proc. SIGMETRICS*, 2009, pp. 37-48.

[7] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," in *Proc. INFOCOM*, 1999, pp. 126-134.

[8] L. Fan, P. Cao, J. Almeida, and A.Z. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Trans. Netw.*, vol. 8, pp. 281-293, June 2000.

[9] H. Che, Z. Wang, and Y. Tung, "Analysis and Design of Hierarchical Web Caching Systems," in *Proc. IEEE INFOCOM*, 2001, pp. 1416-1424.

[10] N. Fujita, Y. Ishikawa, A. Iwata, and R. Izmailov, "Coarse-Grain Replica Management Strategies for Dynamic Replication of Web Contents," *Comput. Netw.*, vol. 45, no. 1, pp. 19-34, 2004.

[11] L. Muscariello, G. Carofiglio, and M. Gallo, "Bandwidth and Storage Sharing Performance in Information Centric Networking," in *ACM SIGCOMM ICN Workshop*, 2011, pp. 26-31.

[12] U. Lee, I. Rimac, and V. Hilt, "Greening the Internet with Content-Centric Networking," in *Proc. eEnergy*, 2010, pp. 179-182.

[13] E.J. Rosensweig and J. Kurose, "Breadcrumbs: Efficient, Best-Effort Content Location in Cache Networks," in *Proc. INFOCOM*, 2009, pp. 2631-2635.

[14] A.A. Jiang and J. Bruck, "Optimal Content Placement for En-Route Web Caching," in *Proc.IEEE NCA*, 2003, pp. 9-16.

[15] P. Krishnan, D. Raz, and Y. Shavitt, "The Cache Location Problem," *IEEE/ACM Trans. Netw.*, vol. 8, no. 5, pp. 568-582, 2000.

[16] S. Arianfar, P. Nikander, and J. Ott, "On Content-Centric Router Design and Implications," in *Proc. ReArch Workshop*, 2010, pp. 1-6.

[17] M. Varvello, D. Perino, and J. Esteban, "Caesar: A Content Router for High Speed Forwarding," in *Proc. SIGCOMM ICN Workshop*, 2012, pp. 73-78. [Online]. Available: http://doi.acm.org/10.1145/2342488.2342505

[18] I. Psaras, R.G. Clegg, R. Landa, W.K. Chai, and G. Pavlou, "Modelling and Evaluation of CCN-Caching Trees," in *Proc. IFIP Netw.*, 2011, pp. 78-91.

[19] I. Psaras, W.K. Chai, and G. Pavlou, "Probabilistic in-Network Caching for Information-Centric Networks," in *Proc. 2nd ACM SIGCOMM ICN Workshop*, Aug. 2012, pp. 55-60.

[20] N. Laoutaris, H. Che, and I. Stavrakakis, "The LCD Interconnection of LRU Caches and its Analysis," *Performance Eval.*, vol. 63, no. 7, pp. 609-634, July 2006.

[21] W.K. Chai, N. Wang, I. Psaras, G. Pavlou, C. Wang, G.G. de Blas, F.J. Ramon-Salguero, L. Liang, S. Spirou, A. Beben, and E. Hadjioannou, "Curling: Content-Ubiquitous Resolution and Delivery Infrastructure for Next-Generation Services," *IEEE Commun. Mag.*, vol. 49, no. 3, pp. 112-120, Mar. 2011.

[22] P. Jokela, A. Zahemszky, C.E. Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: Line Speed Publish/Subscribe Inter-Networking," in *Proc. SIGCOMM*, 2009, pp. 195-206.

[23] G. Xylomenos, X. Vasilakos, C. Tsilopoulos, V. Siris, and G. Polyzos, "Caching and Mobility Support in a Publish-Subscribe Internet Architecture," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 52-58, July 2012.

[24] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl, "Network of Information (NetInf) an Information-Centric Networking Architecture," *Comput. Commun.*, vol. 36, no. 7, pp. 721-735, Apr. 2013.

[25] I. Poese, B. Frank, B. Ager, G. Smaragdakis, and A. Feldmann, "Improving Content Delivery Using Provider-Aided Distance Information," in *Proc. 10th ACM SIGCOMM IMC*, 2010, pp. 22-34.

[26] 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, 2011.

[27] Y. Wang, Z. Li, G. Tyson, S. Uhlig, and G. Xie, "Optimal Cache Allocation for Content-Centric Networking," in *Proc. ICNP*, 2013, pp. 1-10.

[28] Icarus: A Simulator for icn Cache Networks 2013. [Online]. Available: http://www.ee.ucl.ac.uk/lsaino/software/icarus
[29] A.L. Barabasi and R. Albert, ''Emergence of Scaling in Random Networks,'' *Science*, vol. 286, no. 5439, pp. 509-512, Oct. 1999.
[30] D. Rossi and G. Rossini, ''Caching Performance of Content Centric Networks Under Multi-Path Routing,'' Telecom ParisTech, Paris, France, Tech. Rep., 2011.
[31] S. Bhattacharjee, K.L. Calvert, and E.W. Zegura, ''Self-Organising Wide-Area Network Caches,'' in *Proc. IEEE INFOCOM*, 1998, pp. 600-608.
[32] T. Lauinger, N. Laoutaris, P. Rodriguez, T. Strufe, E. Biersack, and E. Kirda, ''Privacy Risks in Named Data Networking: What is the Cost of Performance?'' Proc. SIGCOMM CCR, 2012, vol. 42, pp. 54-57.

Ioannis Psaras received a diploma in Electrical and Computer Engineering from Democritus University of Thrace, Greece in 2004, and the PhD degree from the same institute in 2008. He won the Ericsson Award of Excellence in Telecommunications for his diploma dissertation in 2004. Ioannis has worked, as a research intern at DoCoMo Eurolabs (May-September 2005) and at Ericsson Eurolab (May-September 2006). His research interests are in the areas of Congestion/Flow Control, Transport-layer Protocols, Information-Centric Networks, Delay-/Disruption-Tolerant Networks (DTNs), User-Provided and User-Centric Networks. He is currently working as senior research associate at the Electronic and Electrical Engineering Department of University College London (UCL). Further information on current and previous research projects he has been involved in can be found at http://www.ee.ucl.ac.uk/~uceeips/. He is a member of the IEEE.

Wei Koong Chai was awarded the BEng degree in Electrical Engineering from the Universiti Teknologi Malaysia, Malaysia in 2000 and both the MSc (Distinction) and the PhD degrees from University of Surrey, UK, in 2002 and 2008, respectively. He is currently a senior research associate at the Department of Electronic and Electrical Engineering, University College London, UK. His research spans across heterogeneous networks including wired/wireless networks and cyber physical systems such as smart grid. His current research interests include information-centric networking (ICN), smart grid communication, quality of service (QoS), resource management (e.g., for satellite networks and wireless mesh networks), cross-layer design (specifically on interaction of protocols at different layers), traffic engineering and network optimization. He is a member of the IEEE.

George Pavlou is Professor of Communication Networks in the Department of Electronic and Electrical Engineering, University College London, UK where he coordinates research activities in networking and network management. He received a Diploma in Engineering from the National Technical University of Athens, Greece and MSc and PhD degrees in Computer Science from University College London, UK. His research interests focus on networking and network management, including aspects such as traffic engineering, quality of service management, policy-based systems, autonomic networking, information-centric networking and software-defined networks. He has been instrumental in a number of European and UK research projects that produced significant results with real-world uptake and has contributed to standardization activities in ISO, ITU-T and IETF. He has been the technical program chair of several conferences and in 2011 he received the Daniel Stokesbury award for ''distinguished technical contribution to the growth of the network management field''. He is a senior member of the IEEE.